FULL-FORMULATION DC-OPF SYNTHESIS: A PROBE INTO HOW LLMS FAIL IN PHYSICAL MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce full-formulation direct-current optimal power flow (DC-OPF) synthesis as a probe of how reliably large language models (LLMs) construct mathematical models of physical systems. The task requires an LLM to construct a complete executable formulation from a natural-language dispatch instruction and an explicit modeling contract, without a supplied OPF backbone. We construct a benchmark of 118 instructions over three PGLib systems and develop a certificate-based evaluator for instance-level formulation equivalence. Its bidirectional violation linear programs detect discrepancies in hard feasible regions even when candidate and reference optimal values agree. We precisely specify the mathematical objects assessed by the evaluator and the scope of certification. Across nine LLMs, 244 of 1,062 outputs fail the certificate checks. A source-level fidelity review of all 244 failed outputs identifies substantive modeling errors in every evaluated model. These findings motivate a different allocation of modeling responsibilities: the LLM interprets the operational meaning of a natural-language requirement, while deterministic software binds it to concrete network entities and instance-dependent data and constructs the complete DC-OPF. A protocolized workflow implementing this allocation raises aggregate certificate passes from 818/1,062 to 1,054/1,062.

1 INTRODUCTION

A recent expert audit suggests that scores on closed-ended physics benchmarks can substantially understate frontier-model performance because apparent failures may arise from benchmark or grading defects (Ansari et al., 2026). We ask a complementary question: beyond obtaining correct final answers, can large language models (LLMs) reliably construct complete executable mathematical models of physical systems? We use direct-current optimal power flow (DC-OPF) as a controlled probe of this capability.

We study *full-formulation DC-OPF synthesis*, in which an LLM constructs a complete executable formulation from a natural-language dispatch instruction and an explicit modeling contract, without a supplied OPF backbone. DC-OPF is widely used in power-system operations (Yang et al., 2018; Stott et al., 2009) and provides a tractable but nontrivial starting point: its convex structure permits deterministic analysis (Boyd & Vandenberghe, 2004), while model construction requires consistent units, sign conventions, and reference directions—requirements also common in other physical-modeling settings.

Recent work has begun examining LLMs on OPF modeling tasks. ProOPF asks LLMs to apply parameter adjustments and structural extensions to a supplied canonical OPF backbone, which already encodes the underlying physical relations (Shen et al., 2026). Our task directly tests the ability to construct these relations, exposing modeling errors that a supplied backbone can mask. Figure 1 illustrates the distinction through the analogy of modifying an existing building and constructing one from its specifications.

Evaluating this capability requires both checking the complete formulation and stating precisely what the evaluator certifies. Existing optimization-modeling benchmarks commonly compare structured targets or numerical outcomes (Ramamonjison et al., 2022; Yang et al., 2025), but a generated formulation may share an optimal solution with the reference while omitting or misencoding non-binding safety constraints that restrict other operating points. The automated evaluator therefore tests instance-level formulation equivalence, including equality of candidate and reference hard

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.

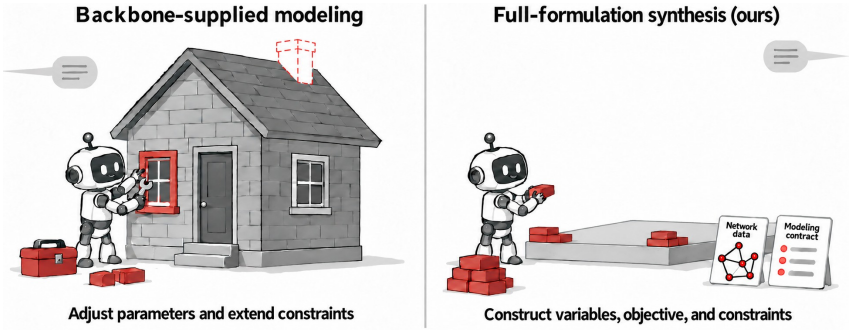


Figure 1: **Formulation responsibilities in the two settings.** A supplied OPF backbone supports parameter adjustments and constraint extensions (left). Full-formulation DC-OPF synthesis requires constructing all variables, the objective, and all constraints from the public inputs (right).

feasible regions. The evaluator assesses equivalence of the resulting mathematical objects on the evaluated instance, rather than whether the program faithfully implements every prescribed modeling requirement.

Our experiments reveal substantive physical-modeling errors in the evaluated LLMs, motivating a different allocation of modeling responsibilities. The LLM interprets the operational meaning of each natural-language requirement. Reusable deterministic software then binds the interpreted requirement to the network instance and constructs the complete DC-OPF.

The contributions are fourfold.

1. We formalize full-formulation DC-OPF synthesis to evaluate complete physical-model construction, directly testing LLMs' ability to build the physical relations otherwise supplied by an OPF backbone. We construct a benchmark of 118 instructions over three PGLib systems (Babaeinejadsarookolaee et al., 2019), covering common continuous single-period DC-OPF operational requests.
2. We develop a certificate-based evaluator for a precisely scoped notion of instance-level formulation equivalence, using bidirectional violation linear programs (LPs) to test mutual containment of hard feasible regions, and distinguish this automated criterion from the stronger notion of full-formulation fidelity.
3. We evaluate nine LLMs on full-formulation DC-OPF synthesis and conduct a fidelity review of all 244 outputs that fail automated certificate checks. Every evaluated model exhibits substantive physical-modeling errors. We organize the reviewed failures into four first-error categories: physical relations, entity binding and scope, constraint semantics, and program implementation.
4. We propose an allocation of modeling responsibilities in which the LLM interprets natural-language requirements, while deterministic software binds and constructs the complete DC-OPF. We implement and evaluate this allocation through a protocolized workflow. Aggregate certificate passes increase from 818/1,062 under full-formulation synthesis to 1,054/1,062 under the proposed allocation.

2 Related Work

LLMs for physical understanding and reasoning. PhysReason evaluates physics problem solving through final answers and intermediate solution steps (Zhang et al., 2025b). SciCode evaluates scientific code generation through numerical and domain-specific tests (Tian et al., 2024). PhysCodeBench evaluates generated physics simulation programs through physical assertions and conservation checks (Xie et al., 2026). We use DC-OPF as a controlled setting to evaluate whether LLMs can construct complete executable mathematical models of a physical system, assessing the resulting formulations through objective agreement and hard feasible-region equivalence.

LLMs for power-system operation and optimization. Recent work studies domain expertise, control, and optimization in power systems. ElecBench evaluates power-system knowledge (Zhou et al., 2024), and GAIA develops a domain model for power dispatch (Cheng et al., 2025). Grid-Agent combines control-action planning with tool-based validation (Zhang et al., 2025c). ProOPF is the most directly related benchmark, translating natural-language requirements into parameter adjustments and structural extensions to a canonical OPF formulation (Shen et al., 2026). Full-formulation DC-OPF synthesis tests complete formulation construction from dispatch instructions, including both the physical relations supplied by ProOPF's backbone and instruction-specific changes under an explicit modeling contract.

LLMs for optimization modeling and formulation verification. Prior work develops semantic parsers (Ramamonjison et al., 2022; Ahmed & Choudhury, 2024) and optimization-modeling systems and benchmarks (Huang et al., 2025b; AhmadiTeshnizi et al., 2024; Yang et al., 2025; Huang et al., 2025a; Jiang et al., 2025). Data synthesis, structured validation, and solver-informed learning improve modeling capability (Lu et al., 2025; Wu et al., 2025; Chen et al., 2025). Text2Opt-Bench studies instance-data binding as a bottleneck in optimization modeling, evaluating generated programs through execution and objective-value agreement (Gao et al., 2026). EquivaMap discovers variable mappings with LLMs and deterministically checks mapped solutions for feasibility and optimality (Zhai et al., 2025). Concurrent ModelEquivBench uses Farkas containment certificates, dual to our violation LPs for aligned continuous polyhedra (Zhu & Xu, 2026). Its variable-alignment search faces combinatorial growth: exhaustive permutation coverage is declared only for $n \leq 7$ variables without binary coordinates, limiting conclusive non-equivalence judgments beyond this regime. Our modeling contract fixes coordinate correspondence through public physical identities, eliminating this search and enabling complete feasible-region comparisons on the evaluated 30-, 118-, and 300-bus systems.

Allocation of modeling responsibilities. OptiMUS and OR-LLM-Agent divide formulation, programming, and debugging among specialized LLM components (AhmadiTeshnizi et al., 2024; Zhang et al., 2025a). MURKA combines structured extraction with LLM-generated models and iterative verification (Xie et al., 2025). Hu et al. (2025) combine LLM-based numerical parameter extraction and model generation with validation and iterative repair. For active distribution network (ADN) dispatch, Yang et al. (2026) provide domain modeling knowledge to support LLM-based model construction. The proposed allocation assigns instruction interpretation to the LLM and instance grounding and construction of the DC-OPF physical model to deterministic components. These components read and update numerical network data and compute the coefficients used in the formulation.

3 Full-Formulation DC-OPF Synthesis

Full-formulation DC-OPF synthesis requires realizing instruction-induced state changes and constructing all variables, the objective, and all physical and operational constraints of a single-period continuous DC-OPF without invoking the solver. We use *full-formulation fidelity* for faithful implementation of these requirements; the evaluator assesses *instance-level formulation equivalence*.

3.1 Problem Formulation and Task Definition

Quadratic programs with soft constraints. We consider convex quadratic programs (QPs) with prescribed affine soft constraints:

$$\begin{aligned} \min_{x,s} \quad & \tfrac{1}{2} x^\top Q x + c^\top x + c_0 + \lambda^\top s \\ \text{s.t.} \quad & Ax \leq b, \qquad Ex = h, \\ & (x, s) \in \mathcal{H}, \qquad s \geq 0. \end{aligned} \tag{1}$$

Here $x \in \mathbb{R}^n$ contains the public decision quantities, $Q = Q^\top \succeq 0$, and s contains the prescribed soft slacks. We denote the objective by $f(x, s)$. The hard affine rows include variable bounds and fixed values. For an affine residual $h_k(x)$, $\mathcal{H}$ prescribes the epigraph $h_k(x) \leq s_k$ for a soft inequality or $-s_k \leq h_k(x) \leq s_k$ for a soft equality, with linear penalty coefficient λ_k. Two-sided limits use separate one-sided residuals and slacks.

DC-OPF as an instantiation. On a network snapshot, DC-OPF with soft constraints instantiates equation 1 with $x = (P, \theta)$, collecting generation in MW and bus angles in radians. The objective and constraints are determined by the snapshot, instruction, and modeling contract. Appendix C gives the complete formulation of DC-OPF.

Task and context for the candidate LLM. For each natural-language dispatch instruction, the candidate LLM generates code that applies the requested network-state changes and constructs the complete DC-OPF. The accompanying public modeling contract describes the data fields that may be used for modeling, together with their physical meanings and units. It also provides the minimal information needed to uniquely specify the intended DC-OPF variant. The LLM receives the instruction and contract during code generation; numerical values are supplied to the generated code at execution through an independent network snapshot for each task. The LLM has no access to these instance-specific numerical network values. Appendix B gives further details about the context.

Reducing software burden. Full-formulation synthesis still requires executable Python/Pyomo code, but we reduce interface-specific friction by using ordinary nested dictionaries and generic, non-domain Pyomo examples. Compared with a DataFrame-backed interface and minimal Pyomo guidance on the same 118 tasks, build/static failures fall from 15/118 to 2/118 and candidates reaching semantic evaluation increase from 103/118 to 116/118 (Appendix D.3). Software implementation nevertheless remains a genuine part of the end-to-end task: 69/244 formal failures in the subsequent nine-model evaluation are attributed to program implementation under first-error attribution.

3.2 Benchmark Construction

We construct 118 English instructions over the case30, case118, and case300 PGLib systems. PGLib provides publicly available benchmark power-system cases. The benchmark contains 102 feasible tasks, six infeasible tasks, and ten expected-rejection tasks. The task set covers a broad range of common continuous single-period DC-OPF operational requests under the modeling contract, including network and parameter changes, generator coordination, reserve requirements, interface and angle constraints, and hard/soft security requirements. Instructions are phrased in domain language rather than as schema-level commands. All benchmark instructions and their associated task specifications were reviewed by two power-system experts for technical correctness and consistency with the modeling contract.

4 Certificate-Based Evaluation

We describe the evaluator through the structured problem in equation 1, instantiated by the public DC-OPF interface. An incorrect formulation can share an optimum with the reference while admitting extra points or excluding other valid points. We therefore compare the transformed state, objective and soft epigraphs, and complete hard feasible regions, using bidirectional violation LPs to test whether either formulation admits points excluded by the other.

Why matching objective values is insufficient. In `case30_A_029`, an MVA-base conversion error in GPT 5.6 Luna's formulation changes the network equations. The candidate-declared and externally reconstructed costs both match the reference optimum at $8{,}036.3056$ within tolerance. However, bidirectional violation LPs expose the incorrect candidate feasible region: each formulation admits points violating the other's nodal balance. Among the 252 formally rejected candidates in Section 6, violation LPs detect 26 erroneous formulations, including 11 for which the candidate-declared objective value, externally reconstructed objective value, and reference optimum all match.

4.1 Equivalence criteria and evaluation pipeline

Subscripts C and R denote candidate and reference quantities, respectively. For feasible tasks, we define *instance-level formulation equivalence* through three conditions: (i) **Equal transformed states** rule out candidates that modify the network differently. (ii) **Coefficient-wise objective agreement**, $f_C(x, s) \equiv f_R(x, s)$, ensures identical objective functions. Additional checks of the prescribed soft epigraphs and penalties rule out incorrect soft-constraint formulations. (iii) **Equal hard feasible**

regions, $\mathcal{F}_C = \mathcal{F}_R$, rule out candidates admitting different operating points. The definition permits different Pyomo names, constraint orders, and algebraic representations that preserve these conditions.

Figure 2 summarizes the semantic checks. The evaluator applies these checks within explicit numerical tolerances. Infeasible tasks require state and objective checks to pass, together with solver-confirmed infeasibility; expected-rejection tasks must raise the prescribed exception. Reference construction, evaluator validation, and numerical settings are detailed in Appendix J.

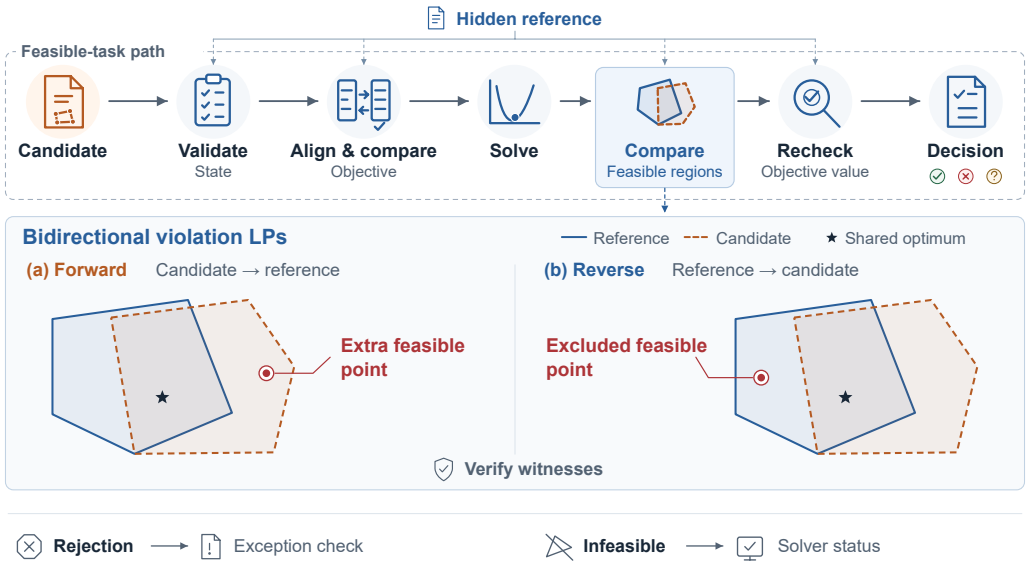


Figure 2: **Certificate-based evaluation and bidirectional violation LPs.** The top path groups the semantic checks for feasible tasks; objective alignment includes the soft-penalty constraints. The bottom row covers other outcomes. A shared optimum (black star) can coexist with (a) extra points admitted by the candidate or (b) reference-feasible points excluded by the candidate. Red witnesses are verified in their source feasible regions.

4.2 Bidirectional feasible-region certification

For $X \in \{C, R\}$, we normalize variable bounds, fixed values, and hard affine constraints in the shared public coordinates $x \in \mathbb{R}^n$, aligned by their semantic identities after the state check. The hard feasible region is

$$\mathcal{F}_X = \left\{ x : r_j^X(x) \le 0,\ j \in \mathcal{I}_X; \quad r_j^X(x) = 0,\ j \in \mathcal{E}_X \right\}, \tag{2}$$

where $r_j^X(x)$ denotes the affine constraint residual for normalized hard constraint j, and $\mathcal{I}_X$ and $\mathcal{E}_X$ index the hard inequalities and equalities, respectively. Soft constraints involving s_k are checked with the objective. The containment tests use complete affine descriptions of both hard feasible regions in x.

To apply the mutual-containment test for polyhedra, we solve the following violation LP over each source feasible region $\mathcal{F}_X$ for each target constraint j with constraint residual $r_j^Y(x)$:

$$V_{X\to Y}(j, d) = \max_{x \in \mathcal{F}_X} d\, r_j^Y(x), \qquad d \in \begin{cases} \{+1\}, & j \in \mathcal{I}_Y, \\ \{+1, -1\}, & j \in \mathcal{E}_Y. \end{cases} \tag{3}$$

Fig. 2(a) illustrates the forward tests $V_{C\to R}$, which maximize violations of the reference constraints over $\mathcal{F}_C$. Passing all tests establishes $\mathcal{F}_C \subseteq \mathcal{F}_R$; a decisive violation yields a witness $x^\dagger \in \mathcal{F}_C \setminus \mathcal{F}_R$ showing that the candidate admits a point excluded by the reference.

Fig. 2(b) illustrates the reverse tests $V_{R\to C}$, which maximize violations of the candidate constraints over $\mathcal{F}_R$. Passing all tests establishes $\mathcal{F}_R \subseteq \mathcal{F}_C$; a decisive violation yields a witness in $\mathcal{F}_R \setminus \mathcal{F}_C$ showing that the candidate excludes a reference-feasible point. For the nonempty polyhedra of

feasible tasks,

$$\mathcal{F}_C = \mathcal{F}_R \quad \Longleftrightarrow \quad \mathcal{F}_C \subseteq \mathcal{F}_R \ \wedge \ \mathcal{F}_R \subseteq \mathcal{F}_C. \tag{4}$$

The criterion permits positive constraint scaling, exchanging sides, constraint reordering, two-sided constraint expansion, and redundant representations preserving the hard feasible region. With m_R and m_C normalized hard constraints in these coordinates, the procedure solves $O(m_R + m_C)$ violation LPs and is polynomial in normalized model size.

Alignment in public physical coordinates. The candidate model must provide public-ID maps for generator active-power outputs P and bus voltage angles θ, together with a map of the prescribed soft-constraint auxiliaries s; these exhaust its variables. With arbitrary auxiliaries, reverse containment becomes containment of the reference region in the candidate's projection onto $x = (P, \theta)$, a problem that is co-NP-complete for rational polytopes in variable dimension (Kellner, 2015, Theorem 2.4), which is further discussed in Appendix J.2.

4.3 Interpretation and limitations

For feasible tasks, certificate checks assess instance-level formulation equivalence; fidelity concerns implementation of prescribed requirements. Zero-effect errors leave the compared objects unchanged.

A zero-effect error on case30. On frozen case30, thermal limits and branch-flow relations for all 41 active branches imply $|\theta_{i_a} - \theta_{j_a}| \leq F_a^{\max}/(S_{\text{base}}|\beta_a|)$. The largest bound is 0.30831472 rad, approximately $17.665132°$, strictly below $30°$. Omitting the prescribed $\pm 30°$ angle-difference constraints therefore preserves the entire hard feasible region.

Separation of equivalence and fidelity. We conduct a complete fidelity audit of DeepSeek V4 Flash's 236 outputs, comprising 118 from full-formulation synthesis and 118 from the protocolized workflow. Among the 219 outputs that pass the certificate checks, 5/219 (2.28%) are unfaithful, all from full-formulation synthesis.

Also, we find that all generator quadratic-cost coefficients satisfy $c_{2g} = 0$ in the three network cases. This increases the possibility of zero-effect errors, since proportional updates to zero produce no distinguishable mathematical change. We therefore audit whole-task fidelity for 100 task outputs involving c_{2g} updates. Among the 77 outputs that pass the certificate checks, 11/77 (14.29%) are unfaithful. Appendix I discusses this separation in greater detail.

5 Responsibility Allocation for DC-OPF Physical Modeling

In the DC-OPF setting studied here, physical modeling comprises interpreting the operational meaning of a natural-language requirement, binding its scopes and references to concrete network entities and instance-dependent data, and constructing the complete DC-OPF. Construction builds the physical relations, instantiates the constraints and objective, and implements the resulting model in Python/Pyomo.

Responsibility allocation. Full-formulation DC-OPF synthesis assigns all three responsibilities to the LLM and its generated code. Binding and construction follow recurring domain and interface rules, making their implementations reusable. At the same time, both operations are highly sensitive to errors introduced by the LLM. Interpretation, however, must accommodate varied language and dispatch semantics. We therefore assign interpretation to the LLM and binding and construction to reusable deterministic software (Figure 3).

Protocolized implementation. The allocation is implemented through a protocol interface comprising a structured dispatch specification and an entity-explicit form. In the structured dispatch specification, reusable entity sets and delayed references let the LLM express intent compactly without repeated entity enumeration or access to concrete network values, while custom affine constraints over permitted DC physical quantities preserve instruction-specific expressiveness within the protocol's scope. Deterministic software uses a compiler to expand the specification into the entity-explicit form, thereby binding its scopes and references to concrete entities; it then applies the specified

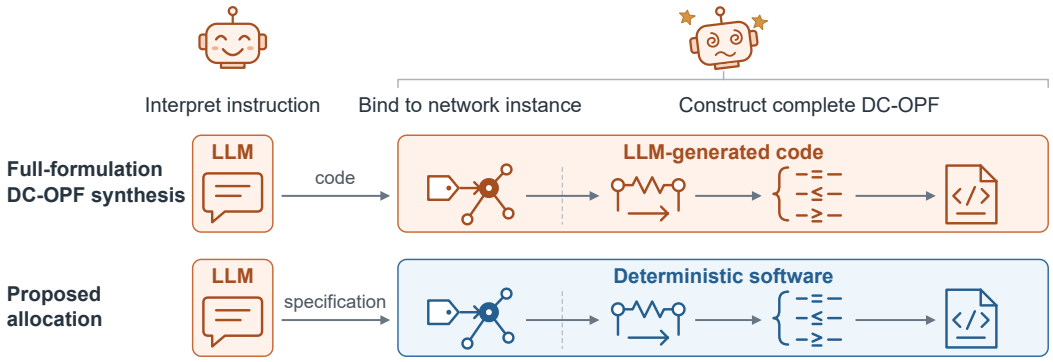


Figure 3: **Allocation of modeling responsibilities.** Both paths return a transformed network state and a DC-OPF model. The proposed allocation assigns network-instance binding and complete DC-OPF construction to reusable deterministic software.

changes and uses predefined logic together with a human-authored DC-OPF component library to assemble the complete optimization problem. Across 12 tasks selected for mechanism coverage, the production and independently reconstructed formulations agreed on normalized affine-constraint coefficients to within 2.22×10^{-16} (Appendix K). The workflow uses four in-context examples. In a controlled comparison with DeepSeek V4 Flash, removing these examples while holding all other prompt content and task definitions fixed reduces certificate passes from 118/118 to 115/118 (Appendix H.4).

6 Results

6.1 Experimental setting

We evaluate nine models from five families on the 118 requests in Section 3.2, using the certificate pipeline in Section 4. The main evaluation uses one LLM call per request. All models use high reasoning effort and no tools, and generate outputs without direct access to the instance's network snapshot. HiGHS solves candidate DC-OPF models and violation LPs through Pyomo APPSI.

6.2 Reliability of full-formulation synthesis

Table 1 reports certificate failures under the two responsibility allocations, with full-formulation failures grouped by first-error category. Across models, full-formulation synthesis passes certificate checks on 53–105 of the 118 tasks, with a median of 96 passes. All models use the same 118 tasks; aggregate counts pool their outputs.

Table 1: Formal failures under the two responsibility allocations (118 tasks per model). Bold marks the lowest model-level total in each setting.

	Full-formulation synthesis					Proposed allocation
Model	Physical relations	Entity binding and scope	Constraint semantics	Program implementation	Total	Total
DeepSeek-V4-Flash-0731	4	5	5	3	17	**0**
DeepSeek-V4-Pro-0813	9	5	5	2	21	**0**
Claude-Opus-5	6	3	1	4	14	1
Claude-Sonnet-5	16	3	8	2	29	1
Gemini-3.1-Pro-Preview	1	8	5	9	23	**0**
Gemini-3.6-Flash	2	7	5	26	40	1
GPT-5.6-Luna	36	12	1	16	65	1
GPT-5.6-Sol	12	4	3	3	22	3
Kimi-K3	4	2	3	4	**13**	1
Aggregate	90	49	36	69	244	8

Errors in full-formulation synthesis. We review the generated source for all 244 full-formulation synthesis outputs that fail certificate checks, identifying substantive modeling errors in every evaluated model.

We group the errors observed when LLMs perform physical modeling for DC-OPF into four categories: physical relations, entity binding and scope, constraint semantics, and program implementation. We count each failure once, assigning it to the category of the earliest confirmed error in the generated source. See Appendix L.1 for the reported error subcategories and their assignments to these categories.

Representative errors include:

1. **MW/per-unit conversion errors (38/244, 15.57%).** In `case118_A_007`, Claude Sonnet 5 omits S_{base} from the required MW flow relation $P_a(\theta) = S_{\text{base}}\beta_a(\theta_{i_a} - \theta_{j_a})$, using per-unit flow in MW-valued soft thermal constraints. All 372 associated soft epigraph constraints disagree with the reference.
2. **Writing dispatch conditions into public state (35/244, 14.34%).** In `case118_A_051`, DeepSeek V4 Flash expresses a dispatch requirement $P_{22} = 20$ MW by changing the minimum, maximum, and current active-power fields. In `case118_A_052`, eight of nine candidates encode a requirement to keep a generator online and dispatch at least 50 MW by writing 50 MW into its minimum-output state, instead of imposing $P_g \geq 50$.
3. **Branch-flow direction or incidence-sign errors (18/244, 7.38%).** In `case300_A_044`, GPT 5.6 Sol forms the angle difference from the public source i_a to destination j_a, then applies an endpoint alignment sign. A branch whose stored endpoint order opposes the public flow direction is consequently reversed twice.

Remaining modeling errors in the evaluated high-end LLMs. System/element MVA base confusion in equivalent-impedance conversion accounts for nine of GPT 5.6 Sol's 22 formal failures across all three network cases. Claude Opus 5 still omits angle-difference safety constraints $\Delta_a^{\min} \leq \theta_{i_a} - \theta_{j_a} \leq \Delta_a^{\max}$ or misencodes branch-flow directions or nodal-balance signs.

Effect of responsibility allocation. We evaluate the proposed allocation through the protocolized workflow, using the same network instances, reference artifacts, certificate pipeline, and one-call budget as full-formulation synthesis. As in full-formulation synthesis, the workflow LLM has no access to instance-specific numerical network values.

For each LLM, the protocolized implementation yields 12–64 additional certificate passes. The aggregate increases from 818/1,062 to 1,054/1,062, and model–task certificate failures fall from 244 to eight, a reduction of 96.72%. Workflow counts range from 115/118 to 118/118 across models, with a median of 117/118. The lowest workflow count exceeds the highest full-formulation count. The eight failures under the proposed allocation comprise three responses without the required structured specification, two syntactically invalid JSON outputs, and three outputs that encode soft requirements as hard constraints.

The higher certificate pass rates and fidelity review findings jointly support the effectiveness of the proposed allocation in improving DC-OPF modeling reliability. Three-run comparisons for GPT-5.6-Luna and Gemini-3.6-Flash preserve the pass-count gap between allocations (Appendix H.5).

6.3 Additional inference-time computation

Holding DeepSeek V4 Flash fixed, we compare single-call full-formulation synthesis, two three-call strategies—critique-and-rewrite reflection and two independent builders followed by an integrator—and the proposed workflow. Table 2 reports certificate pass counts and inference tokens across all 118 requests.

For the evaluated model and benchmark, reflection and the two-builder strategy increase certificate passes from 101/118 to 113/118 and 112/118, using 2.63 and 2.84 times as many inference tokens as single-call synthesis, but leave five and six requests without a passing certificate, respectively.

Remaining failures include accepting operations on devices outside the active-power decision domain, fabricating electrical parameters for a new line, and extending generator updates to synchronous

Table 2: Certificate pass counts and inference tokens for full-formulation synthesis strategies and the proposed workflow. Token accounting is detailed in Appendix D.4.

Method	Calls/task	Certificate passes	Inference tokens
Single-call synthesis	1	101/118	7,269,528
Reflection	3	113/118	19,114,560
Two builders + integrator	3	112/118	20,653,542
Proposed workflow	1	**118/118**	**1,063,746**

condensers (Appendix D.5). The two tested strategies therefore improve reliability but do not eliminate errors in LLM-generated physical realization.

6.4 Intermediate responsibility allocations

We compare two intermediate baselines that progressively transfer modeling responsibilities to deterministic components. The primitive helper provides per-entity physical primitives that the LLM assembles into a complete optimization model. The constraint helper automatically constructs the standard DC-OPF constraints and objective; the LLM configures these constraints and expresses nonstandard affine constraints as needed. The proposed allocation further transfers entity binding to deterministic components. Table 3 reports DeepSeek V4 Flash and GPT 5.6 Luna results. We select Luna because it has the most full-formulation failures among the nine models (65/118; Table 1).

Table 3: Certificate passes across separate runs on 118 tasks. Full-formulation and workflow counts are from Table 1.

Allocation	DeepSeek V4 Flash	GPT 5.6 Luna
Full-formulation synthesis	101/118	53/118
Full-formulation prompt + primitive helper	102/118	91/118
Full-formulation prompt + constraint helper	110/118	103/118
Proposed allocation	118/118	117/118

The primitive helper raises the certificate pass counts from 101/118 to 102/118 for Flash and from 53/118 to 91/118 for Luna; transferring complete model construction to the constraint helper raises them further to 110/118 and 103/118. Across the two models, the combined first-error count in physical relations, constraint semantics, and program implementation decreases from 65 under full-formulation synthesis to 25 with the primitive helper and one with the constraint helper. Entity-binding and scope errors instead remain at 17, 18, and 22, respectively. These results support assigning both instance-dependent binding and complete DC-OPF construction to deterministic software. Appendices H and H.3 give the helper definitions and detailed error diagnoses.

7 Conclusion

We introduce full-formulation DC-OPF synthesis to probe LLMs' physical-modeling capabilities. Certificate-based evaluation and a fidelity review confirm substantive modeling errors in 244 outputs across all nine evaluated LLMs. Separating responsibilities between LLMs and deterministic software improves reliability on this benchmark, reducing aggregate certificate failures from 244/1,062 to 8/1,062.

Two limitations qualify these conclusions. First, the automated evaluator assesses instance-level formulation equivalence under the public contract and numerical tolerances, not full-formulation fidelity. It can miss zero-effect errors. Direct fidelity review evaluates full-formulation fidelity through line-by-line inspection, but it is costly and does not scale. Second, the observed failures are conditional on the evaluated generation context and on DC-OPF as the probe. Although consistent units, reference directions, and faithful constraint encoding are requirements shared by many physical-modeling tasks, the prevalence and form of these errors may differ in other contexts, like a greater software-implementation burden or prompts designed to guard against particular failure modes.

AI ASSISTANCE

We used large language models to assist with implementing the protocolized workflow and experimental code. The authors supplied the explicit mathematical forms and rules for the underlying physical components, including OPF equations, direction and sign conventions, entity binding rules, and power transfer distribution factor (PTDF) matrix construction. For these components, LLMs only implemented the supplied specifications, and the authors reviewed the resulting code. LLMs also helped rephrase existing natural-language benchmark instructions to improve clarity and naturalness while preserving their task semantics and expected outcomes.

We used GPT-6-Astra to assist with code auditing. For all 252 outputs in the main comparison that did not pass the certificate checks, GPT-6-Astra identified the specific failure causes and the applicable evaluation criteria. Human reviewers checked these findings against the automated evaluation results, including the recorded first-error types, and verified the reported causes and consequences before accepting the findings. For certificate-passing outputs included in the fidelity reviews, we used 2 complementary review tracks. A GPT-6-Astra-only track conducted a line-by-line audit of each implementation. Separately, a human–GPT-6-Astra track focused on the dataset's known zero-effect error modes: omitted prescribed constraints that were implied by retained constraints and out-of-scope modifications to zero-valued fields. GPT-6-Astra located the relevant implementation sites, and human reviewers examined those sites. After both tracks were completed, human reviewers made the final fidelity judgments, including judgments on both known and previously unknown zero-effect errors identified during either track.

We additionally used generative AI tools to assist with drafting the manuscript, improving readability, searching for relevant literature, and creating figures. We did not use generative AI for other applicable tasks requiring disclosure under the ICLR 2027 policy. The authors take responsibility for the final content of this work, including AI-assisted text, code, figures, and analyses.

REPRODUCIBILITY STATEMENT

The accompanying code and data package contains the frozen 118-task benchmark and network data, model prompts and recorded generation settings, saved candidate code and protocol outputs, and implementations of the workflow and certificate evaluator. Its pinned dependencies and documented commands support offline re-evaluation of saved candidates, result aggregation, and verification of the reported inference-token totals. Section 6.1 describes the experimental setting. Appendices B and E specify the public data and modeling contract and workflow protocols; Appendix J documents reference and evaluator validation, including numerical tolerances; Appendix K documents independent validation of the production JDIMP-to-Pyomo construction path. The package directory `artifacts/paper_ledgers/` contains four complete CSV exports: the 118-task catalog, all 252 main-comparison failure reviews, the paired fidelity review of the 118 DeepSeek V4 Flash synthesis outputs and 118 corresponding workflow outputs, and the whole-task fidelity review of 100 selected quadratic-cost outputs. Each review row is bound to the released candidate and evaluation evidence by file paths and SHA-256 digests.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024. URL `https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html`.

Tasnim Ahmed and Salimur Choudhury. LM4OPT: Unveiling the potential of large language models in formulating mathematical optimization problems. *INFOR: Information Systems and Operational Research*, 62(4):559–572, 2024. doi: 10.1080/03155986.2024.2388452.

Ali Ansari, Haoran Sun, Andy Zeyi Liu, Mark Jabbour, Yongshan Ding, Steven Girvin, Yu He, Sohrab Ismail-Beigi, Aleksander Kubica, Owen D. Miller, Corey O'Hern, Vidvuds Ozolins, David Poland, A. Douglas Stone, Frank C. van den Bosch, Logan Wright, Navid Akbari, Santanu Antu, Kangle Cai, Andrew Calabrese-Day, Mateo Cárdenes Wuttig, Meng Cheng, Barry T. Chiang, Ali

Ghorashi, Shouzhen Gu, Haoyang Huang, Zhibo Kang, Lukas Kienesberger, Hantian Liu, Charles Lomba, Zhongling Lu, Wenchao Ma, Rohin E. McIntosh, Evan McKinney, Ivan Rojkov, Xulei Sun, Yarone Meir Tokayer, Naveen Balaji Umasankar, Mira Varma, Leda Wang, Qimin Wang, Tyler Wang, Haoyu Wei, Jinming Yang, Jinchen Zhao, Sherlock Tingrui Zhao, Qinyuan Zheng, Jay S. Zou, Lucas Baker, Arman Cohan, and John Sous. How good are frontier models at physics? expert re-grading reveals broken evaluations and near-saturation of leading benchmarks. *arXiv preprint arXiv:2609.13009*, 2026. URL https://arxiv.org/abs/2609.13009v1.

Sogol Babaeinejadsarookolaee, Adam Birchfield, Richard D Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, et al. The power grid library for benchmarking AC optimal power flow algorithms. *arXiv preprint arXiv:1908.02788*, 2019. URL https://arxiv.org/abs/1908.02788.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. URL https://web.stanford.edu/~boyd/cvxbook/.

Yitian Chen, Jingfan Xia, Siyu Shao, DongDong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 106027–106069, 2025. doi: 10.52202/085713-3539.

Yuheng Cheng, Huan Zhao, Xiyuan Zhou, Junhua Zhao, Yuji Cao, Chao Yang, and Xinlei Cai. A large language model for advanced power dispatch. *Scientific Reports*, 15(1):8925, 2025. doi: 10.1038/s41598-025-91940-x.

Zhiqi Gao, Albert Ge, Alexander Berenbeim, Nathaniel D. Bastian, and Frederic Sala. Models can model, but can't bind: Structured grounding in text-to-optimization. *arXiv preprint arXiv:2605.21751*, 2026. doi: 10.48550/arXiv.2605.21751. URL https://arxiv.org/abs/2605.21751.

Yunkai Hu, Tianqiao Zhao, and Meng Yue. From natural language to solver-ready power system optimization: An LLM-assisted, validation-in-the-loop framework. *arXiv preprint arXiv:2508.08147*, 2025. doi: 10.48550/arXiv.2508.08147. URL https://arxiv.org/abs/2508.08147.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9OMvtboTJg.

Kai Kellner. Containment problems for projections of polyhedra and spectrahedra. *arXiv preprint arXiv:1509.02735*, 2015. doi: 10.48550/arXiv.1509.02735. URL https://arxiv.org/abs/1509.02735.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40769–40802. PMLR, 2025. URL https://proceedings.mlr.press/v267/lu25o.html.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Chao Shen, Zihan Guo, Xu Wan, Zhenghao Yang, Yifan Zhang, Wenqi Huang, Jie Song, Zongyan Zhang, and Mingyang Sun. ProOPF: Benchmarking and improving LLMs for professional-grade power systems optimization modeling. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2602.03070v5.

Brian Stott, Jorge Jardim, and Ongun Alsaç. DC power flow revisited. *IEEE Transactions on Power Systems*, 24(3):1290–1300, 2009. doi: 10.1109/TPWRS.2009.2021235.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A research coding benchmark curated by scientists. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30624–30650, 2024. doi: 10.52202/079017-0963. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/36850592258c8c41cecdaa3dea5ff7de-Abstract-Datasets_and_Benchmarks_Track.html.

Yang Wu, Yifan Zhang, Yurong Wu, Yuran Wang, Junkai Zhang, and Jian Cheng. Step-Opt: Boosting optimization modeling in LLMs through iterative data synthesis and structured validation. *arXiv preprint arXiv:2506.17637*, 2025. URL https://arxiv.org/abs/2506.17637.

Tianyidan Xie, Peiyu Wang, Jiaxin Hu, Yuyi Qian, Yuxuan Wang, Shenyi Wang, Rui Ma, Yanlun Peng, Lanjun Wang, Ying Tai, Jian Yang, and Zili Yi. PhysCodeBench: Benchmarking physics-aware symbolic simulation of 3D scenes via self-corrective multi-agent refinement. *arXiv preprint arXiv:2604.23580*, 2026. URL https://arxiv.org/abs/2604.23580v2.

Wantong Xie, Yi-Xiang Hu, Jieyang Xu, Feng Wu, and Xiangyang Li. MURKA: Multi-reward reinforcement learning with knowledge alignment for optimization tasks. In *Advances in Neural Information Processing Systems*, volume 38, pp. 34878–34905, 2025. doi: 10.52202/085713-1171.

Xu Yang, Chenhui Lin, Yue Yang, Qi Wang, Haotian Liu, Haizhou Hua, and Wenchuan Wu. Large language model powered automated modeling and optimization of active distribution network dispatch problems. *IEEE Transactions on Smart Grid*, 17(2):952–965, 2026. doi: 10.1109/TSG.2025.3621438.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fsDZwS49uY.

Zhifang Yang, Haiwang Zhong, Qing Xia, and Chongqing Kang. Fundamental review of the OPF problem: Challenges, solutions, and state-of-the-art algorithms. *Journal of Energy Engineering*, 144(1):04017075, 2018. doi: 10.1061/(ASCE)EY.1943-7897.0000510.

Haotian Zhai, Connor Lawless, Ellen Vitercik, and Liu Leqi. EquivaMap: Leveraging LLMs for automatic equivalence checking of optimization formulations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74288–74305. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhai25a.html.

Bowen Zhang, Pengcheng Luo, Genke Yang, Boon-Hee Soong, and Chau Yuen. OR-LLM-Agent: Automating modeling and solving of operations research optimization problems with reasoning LLM. *arXiv preprint arXiv:2503.10009*, 2025a. URL https://arxiv.org/abs/2503.10009.

Xinyu Zhang, Yuxuan Dong, Yanrui Wu, Jiaxing Huang, Chengyou Jia, Basura Fernando, Mike Zheng Shou, Lingling Zhang, and Jun Liu. PhysReason: A comprehensive benchmark towards physics-based reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16593–16615. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.acl-long.811. URL https://aclanthology.org/2025.acl-long.811/.

Yan Zhang, Ahmad Mohammad Saber, Amr Youssef, and Deepa Kundur. Grid-Agent: An LLM-powered multi-agent system for power grid control. *arXiv preprint arXiv:2508.05702*, 2025c. URL https://arxiv.org/abs/2508.05702.

Xiyuan Zhou, Huan Zhao, Yuheng Cheng, Yuji Cao, Gaoqi Liang, Guolong Liu, Wenxuan Liu, Yan Xu, and Junhua Zhao. ElecBench: a power dispatch evaluation benchmark for large language models. *arXiv preprint arXiv:2407.05365*, 2024. URL https://arxiv.org/abs/2407.05365.

Penglin Zhu and Jungang Xu. ModelEquivBench: Certifying multi-relational evaluation of LLM-generated optimization models. *arXiv preprint arXiv:2607.29431*, 2026. doi: 10.48550/arXiv.2607.29431. URL https://arxiv.org/abs/2607.29431.

A Organization of the Appendix

The appendices provide detailed specifications, workflow documentation, and evaluation evidence. The material is organized as follows:

- **Task and benchmark specifications.** Appendix B specifies the public data, synthesis prompt, and returned artifact; Appendix C states the DC-OPF mathematical convention; and Appendix D presents the benchmark composition and task types, followed by the interface and additional-inference comparisons.
- **Protocolized implementation and coverage.** Appendix E continues Section 5 by describing JDIMP, its deterministic expansion into JDEF, and the subsequent state and model construction. Appendix F then presents the complete public runtime, including solution, safety validation, feedback, and reporting. Appendix G examines protocol coverage of the modeling requirements.
- **Responsibility-allocation and workflow comparisons.** Appendix H defines the 2 intermediate helper conditions, their allocation of construction responsibilities, and their residual failures. Sections H.4 and H.5 compare workflow performance with and without the 4 prompt examples and report two additional complete runs for two selected models under both evaluated responsibility allocations, respectively.
- **Evaluation and correctness.** Appendix I distinguishes instance-level formulation equivalence from full-formulation fidelity, explains the automated correctness criterion, and presents the associated output reviews. Appendix J summarizes the validation evidence, states the evaluator's coordinate and auxiliary-variable assumptions, and documents numerical procedures and controlled tests with nonzero quadratic costs (Section J.4). Appendix K independently validates the production construction path from reviewed JDIMP through deterministic compilation and backend processing to a Pyomo model.
- **Failure analysis and released detailed records.** Appendix L summarizes the complete formal-failure review, gives a containment-failure example, and provides paired workflow comparisons. The accompanying release package provides CSV ledgers containing the complete 118-task catalog, all 252 main-comparison failure reviews, the paired 236-output DeepSeek V4 Flash fidelity review, and the 100-output quadratic-cost fidelity review.

B Public Data and Modeling Contract

The full-formulation synthesis setting exposes an ordinary nested Python dictionary rather than a power-system toolkit object. The root contains scalar system metadata and six entity tables. Every table is a mapping from backend row keys to row dictionaries; the row key is an access coordinate, not a public entity identifier. Public bus identities are the original MATPOWER bus numbers, whereas public generator and branch identities are the one-based row numbers of the source generator and branch matrices. Fields prefixed by `source_` are immutable provenance. Their unprefixed counterparts describe the current state and are the only quantities changed by an instruction when the contract permits that change.

B.1 Public snapshot

The complete public schema is reported below. Missing values are represented only by Python `None`; all other values are ordinary booleans, integers, floating-point numbers, strings, lists, or dictionaries. A bus is active when `in_service` is true. A generator is active when both its own switch and its connected bus are active, and it belongs to the active-power decision domain exactly when it is not a synchronous condenser. A branch is active when its switch and both endpoint buses are active.

Table 4: Complete public snapshot schema.

Field / key	Type / unit	Semantics
Root fields		
name	str	Source case name; contains no Benchmark task identifier.
sn_mva	positive float / MVA	System power base, equal to MATPOWER baseMVA.
f_hz	positive float / Hz	Nominal system frequency.
benchmark_metadata	dict[str, primitive]	Source, contract, representation, and unit metadata defined below.
bus	dict[int, dict]	Current Buses and source Bus provenance.
gen	dict[int, dict]	Non-reference Generator rows.
ext_grid	dict[int, dict]	Reference Generator rows.
line	dict[int, dict]	Line subtype of Branch.
trafo	dict[int, dict]	Two-winding-transformer subtype of Branch.
impedance	dict[int, dict]	Impedance subtype of Branch.
Benchmark-metadata fields		
loader_contract_name	str	pglib-python-dict-public-net.
loader_contract_version	str	Semantic contract version.
loader_contract_sha256	str	SHA-256 of the semantic contract bundle.
representation	str	nested-python-dict.
source_format	str	matpower.
source_dataset	str	PGLib-OPF.
source_dataset_version	str	v23.07.
source_case_name	str	Original case name.
source_case_sha256	str	SHA-256 of original MATPOWER case bytes.
public_table_manifest_sha256	str	Hash of the ordered public schema.
angle_unit	str	rad, except fields explicitly named _degree.
active_power_unit	str	MW.
reactive_power_unit	str	MVAr.
apparent_power_unit	str	MVA.
Bus fields		
name	str | int | float | None	Label; not a public identifier.
vn_kv	positive float / kV	Current nominal line-to-line voltage.
type	str	Inert element-type metadata.
zone	str | int | float | None	Current zone label.
in_service	bool	Current Bus switch. An inactive Bus and all attached elements are excluded.
geo	plain object or None	Drawing metadata with no electrical effect.
max_vm_pu	float / p.u.	Current voltage-magnitude upper bound; it does not create a voltage-magnitude variable in this DC model.
min_vm_pu	float / p.u.	Current voltage-magnitude lower bound; it does not create a voltage-magnitude variable in this DC model.
pd_mw	float / MW	Current signed aggregate constant-power demand; positive withdraws active power and negative injects fixed active power.
qd_mvar	float / MVAr	Current signed aggregate reactive demand; no active-power DC-model effect.
gs_mw_at_1pu	float / MW	Current Bus shunt-conductance active-power withdrawal at unit voltage.
bs_mvar_at_1pu	float / MVAr	Current Bus shunt-susceptance reactive injection; no active-power DC-model effect.
source_bus_id	int	Original MATPOWER BUS_I; unique public Bus identifier.
source_bus_type	int	Original BUS_TYPE.
source_pd_mw	float / MW	Original PD; not authoritative after modification.
source_qd_mvar	float / MVAr	Original QD; not authoritative after modification.
source_gs_mw_at_1pu	float / MW	Original GS; not authoritative after modification.
source_bs_mvar_at_1pu	float / MVAr	Original BS; not authoritative after modification.
source_area	int	Original BUS_AREA.
source_vm_pu	float / p.u.	Original voltage-magnitude initialization.
source_va_rad	float / rad	Original voltage-angle initialization.
source_base_kv	float / kV	Original BASE_KV.
source_zone	int	Original ZONE.
source_vmax_pu	float / p.u.	Original VMAX.
source_vmin_pu	float / p.u.	Original VMIN.
Generator fields shared by gen and ext_grid		
source_gen_id	positive int	1-based original MATPOWER gen row number; unique across both tables.
source_bus_id	int	Original GEN_BUS, expressed as public Bus identity.
source_gen_status	bool	Original status; not authoritative after modification.
bus	int / backend Bus row key	Current connected Bus.
in_service	bool	Current Generator switch.
is_sync_condenser	bool	True for a zero-active-power-range synchronous condenser; such a row has no active-power decision variable.
controllable	bool	Inert control metadata; not a decision-domain criterion.
p_mw	float / MW	Current active-power setpoint; not the optimized value.
q_mvar	float / MVAr	Current reactive setpoint; no active-power DC-model effect.
min_p_mw	float / MW	Current active-power lower bound.
max_p_mw	float / MW	Current active-power upper bound.
min_q_mvar	float / MVAr	Reactive lower bound; no active-power DC-model effect.
max_q_mvar	float / MVAr	Reactive upper bound; no active-power DC-model effect.
vm_pu	float / p.u.	Voltage setpoint metadata; no voltage-magnitude variable exists.
va_degree	float | None / degree	Reference-angle initialization metadata.
sn_mva	float / MVA	Machine rating/base metadata.
scaling	float	Fixed at 1; does not scale active-power bounds.
slack_weight	float	Reference metadata; not a distributed-reference rule.
type	str | None	Inert Generator type metadata.
name	str | int | float | None	Label; not a public identifier.
cost_c2	float / cost-unit/MW^2	Current convex quadratic cost coefficient.
cost_c1	float / cost-unit/MW	Current linear cost coefficient.

Continued on next page

Table 4: Complete public snapshot schema (continued).

Field / key	Type / unit	Semantics
cost_c0	float / cost-unit	Current constant cost.
source_pg_mw	float / MW	Original PG.
source_qg_mvar	float / MVAr	Original QG.
source_qmax_mvar	float / MVAr	Original QMAX.
source_qmin_mvar	float / MVAr	Original QMIN.
source_vg_pu	float / p.u.	Original VG.
source_mbase_mva	float / MVA	Original MBASE.
source_pmax_mw	float / MW	Original PMAX.
source_pmin_mw	float / MW	Original PMIN.
source_pc1_mw	float | None / MW	Optional original PC1.
source_pc2_mw	float | None / MW	Optional original PC2.
source_qc1min_mvar	float | None / MVAr	Optional original QC1MIN.
source_qc1max_mvar	float | None / MVAr	Optional original QC1MAX.
source_qc2min_mvar	float | None / MVAr	Optional original QC2MIN.
source_qc2max_mvar	float | None / MVAr	Optional original QC2MAX.
source_ramp_agc_mw_per_min	float | None	Optional original RAMP_AGC.
source_ramp_10_mw	float | None / MW	Optional original RAMP_10.
source_ramp_30_mw	float | None / MW	Optional original RAMP_30.
source_ramp_q_mvar_per_min	float | None	Optional original RAMP_Q.
source_apf	float | None	Optional original APF.
source_cost_model	int	Original polynomial-cost model code.
source_startup_cost	float	Original startup cost.
source_shutdown_cost	float	Original shutdown cost.
source_cost_data	list[float]	Original polynomial coefficients; not authoritative after modification.
Fields common to every branch subtype		
source_branch_id	positive int	1-based original MATPOWER branch row number; unique across all subtype tables.
source_from_bus_id	int	Original F_BUS, as public Bus identity.
source_to_bus_id	int	Original T_BUS, as public Bus identity.
source_flow_alignment_sign	int in {−1, +1}	Source-direction flow equals this value times flow in current subtype endpoint order.
source_br_status	bool	Original status; not authoritative after modification.
in_service	bool	Current switch. A Branch is active only when true and both endpoint Buses are active.
rate_a_mw	positive float | None / MW	Current symmetric active-power limit. A finite value bounds source-direction flow to [−rate_a_mw, +rate_a_mw]; None means no finite limit.
angle_min_rad	float | None / rad	Current lower bound on Theta[source_from]−Theta[source_to].
angle_max_rad	float | None / rad	Current upper bound on Theta[source_from]−Theta[source_to].
source_total_charging_b_pu	float / p.u.	Original charging susceptance; no active-power transfer-coefficient effect.
source_rate_b_mva	positive float | None / MVA	Original RATE_B; provenance only.
source_rate_c_mva	positive float | None / MVA	Original RATE_C; provenance only.
source_tap_ratio	positive float	Original off-nominal tap; provenance only under this Benchmark convention.
source_shift_rad	float / rad	Original phase shift; provenance only under this Benchmark convention.
Line-only fields		
name	str | None	Label.
std_type	str | None	Inert materialized type label.
from_bus	int	Current start backend Bus row key.
to_bus	int	Current end backend Bus row key.
length_km	positive float / km	Length.
r_ohm_per_km	float / Ω/km per circuit	Series resistance.
x_ohm_per_km	float / Ω/km per circuit	Series reactance.
c_nf_per_km	float	Shunt capacitance; no active-power coefficient effect.
g_us_per_km	float	Shunt conductance metadata; not used in the DC transfer coefficient.
max_i_ka	float / kA	Equipment rating; not rate_a_mw.
df	float	Derating metadata; not rate_a_mw.
parallel	positive int	Parallel circuit count.
type	str | None	Inert type metadata.
geo	plain object or None	Drawing metadata.
max_loading_percent	float / %	Equipment threshold; not rate_a_mw.
Transformer-only fields		
name	str | None	Label.
std_type	str | None	Inert materialized type label.
hv_bus	int	Current high-voltage endpoint backend Bus row key.
lv_bus	int	Current low-voltage endpoint backend Bus row key.
sn_mva	positive float / MVA per transformer	Element impedance base.
vn_hv_kv	positive float / kV	Rated high-side voltage.
vn_lv_kv	positive float / kV	Rated low-side voltage.
vk_percent	float / %	Current short-circuit impedance magnitude with reactance sign.
vkr_percent	float / %	Current resistance component.
pfe_kw	float	Iron-loss metadata.
i0_percent	float	No-load-current metadata.
shift_degree	float / degree	Current fixed phase-shift metadata; ignored under this Benchmark convention.
tap_side	str | None	Tap metadata.
tap_neutral	number or None	Tap metadata.
tap_min	number or None	Tap metadata.
tap_max	number or None	Tap metadata.
tap_step_percent	float | None	Tap metadata.
tap_step_degree	float | None	Tap metadata.

Continued on next page

Table 4: Complete public snapshot schema (continued).

Field / key	Type / unit	Semantics
tap_pos	number or None	Tap metadata.
tap_changer_type	str | None	Tap metadata.
parallel	positive int	Parallel transformer count.
df	float	Capacity metadata; not rate_a_mw.
max_loading_percent	float	Equipment threshold; not rate_a_mw.
oltc	bool	Tap metadata.
Impedance-only fields		
name	str | None	Label.
from_bus	int	Current start backend Bus row key.
to_bus	int	Current end backend Bus row key.
rft_pu	float / p.u. on sn_mva	Current from-to series resistance.
xft_pu	float / p.u. on sn_mva	Current from-to series reactance.
rtf_pu	float / p.u. on sn_mva	Current to-from series resistance.
xtf_pu	float / p.u. on sn_mva	Current to-from series reactance.
gf_pu	float	From-side shunt conductance; no active-power coefficient effect.
bf_pu	float	From-side shunt susceptance; no active-power coefficient effect.
gt_pu	float	To-side shunt conductance; no active-power coefficient effect.
bt_pu	float	To-side shunt susceptance; no active-power coefficient effect.
sn_mva	positive float / MVA	Element per-unit base.

The current fixed active withdrawal at bus i is $d_i = \texttt{pd_mw}_i + \texttt{gs_mw_at_1pu}_i$. The named total active load, when requested by an instruction, is the signed sum of pd_mw over the requested active buses and does not include the shunt-conductance term. Generator cost is exactly $c_{2g}P_g^2 + c_{1g}P_g + c_{0g}$ using the three current cost fields.

Branch is the union of the line, transformer, and impedance subtype tables. The public source direction is always source_from_bus_id→source_to_bus_id. The current rate_a_mw is an independent symmetric active-power limit; equipment-current or loading metadata does not replace it. The system-base series parameters are derived from the subtype-specific current fields according to Appendix C.

B.2 Prompt for full-formulation DC-OPF synthesis

The full-formulation DC-OPF synthesis prompt combines a fixed system message with one natural-language dispatch instruction. The system message includes the public data contract summarized in Appendix B.1, which specifies the available fields, public identities, units, and electrical semantics. The instruction is available during code generation; execution receives a fresh public snapshot. The requested output is a self-contained Python module containing source code only.

The prompt specifies two entry points. modify_network(net) applies the requested topology and parameter changes while preserving source provenance. build_model(modified_net) constructs the complete, live Pyomo DC-OPF from the updated state and returns the artifact in Appendix B.3. Generic Pyomo examples illustrate sets, variables, constraints, and objectives. The candidate is responsible for entity resolution, active-set construction, physical coefficients, standard DC-OPF relations, and instruction-specific constraints.

The modeling instructions fix the resistance-aware branch convention, reference angle, convex generation cost, and soft-penalty normalization described in Appendix C. The modeling instructions also specify the permitted variables and require model construction without invoking a solver or a prebuilt power-system model. A supported but infeasible request still requires a complete, state-correct model. Invalid or ambiguous requests, unsupported quantities, and topology changes that leave an empty or disconnected active network must raise UnsupportedRequest.

B.3 RETURNED ARTIFACT

Table 5: Required return artifact for full-formulation synthesis.

Item	Requirement
Transformed state	The returned public snapshot realizes exactly the permitted instruction-induced changes.
`model`	A live Pyomo `ConcreteModel` with exactly one active convex minimization objective.
`Pg`	A map from every and only active, non-synchronous-condenser public generator ID to an MW-valued model variable.
`Theta`	A map from every active public bus ID to a radian-valued model variable.
`Soft`	A contiguous zero-based map containing every and only the soft-epigraph variables; it is empty when no soft relation is present.
Variable set	The model variables are exactly the union of `Pg`, `Theta`, and `Soft`; branch-flow variables and hidden auxiliary variables are not part of the interface.
Typed rejection	Unsupported or non-uniquely-groundable requests raise the prescribed exception rather than being silently ignored.

C MATHEMATICAL CONVENTION FOR THE DC-OPF

This section gives the complete DC-OPF instantiation of equation 1, including its default formulation, instruction-specific relations, and base conversions. The convention specifies sets, units, directions, and soft-constraint semantics and is independent of software entry points or reproduction commands.

At the problem level, after applying the state changes requested by a supported instruction, the model has the form

$$\begin{aligned} \min_{P,\theta,s} \quad & f_{\mathrm{DC}}(P,\theta) + \sum_{k\in S} \lambda_k s_k \\ \text{s.t.} \quad & \text{nodal balance, variable bounds, branch limits, angle limits,} \\ & \text{and the reference-angle constraint,} \\ & \text{instruction-specific hard affine relations,} \\ & \text{instruction-specific soft epigraph relations,} \qquad s \geq 0. \end{aligned}$$

Here P, θ, and s collect generation, bus-angle, and soft-epigraph variables, respectively; f_{DC} is the generation-cost objective; S indexes the prescribed soft relations; and each λ_k is a prescribed soft-violation penalty. The following subsections define the sets, variables, branch coefficients, and base conversions; instantiate the default physical relations, limits, and objective; and then add the instruction-specific hard and soft relations.

C.1 SETS, VARIABLES, AND BRANCH COEFFICIENTS

Let B, G, and A denote the active buses, active dispatchable generators, and active branches after the requested state transformation. Synchronous condensers do not belong to G. Let G_i be the generators connected to bus i, and let A_i^+ and A_i^- contain branches whose public source and destination, respectively, are bus i. Active power P_g is measured in MW and bus angle θ_i in radians. The unique reference bus i_{ref} satisfies $\theta_{i_{\mathrm{ref}}} = 0$.

For a source-oriented branch $a : i_a \to j_a$, let r_a and x_a be the per-unit series resistance and reactance on the system power base. The branch transfer coefficient and flow are

$$\beta_a = \frac{x_a}{r_a^2 + x_a^2}, \qquad P_a(\theta) = S_{\mathrm{base}}\beta_a(\theta_{i_a} - \theta_{j_a}). \tag{5}$$

Here S_{base} is the system power base in MVA. The reverse-direction flow is $-P_a(\theta)$. Resistance therefore affects the transfer coefficient through β_a, while active-power losses, charging, fixed tap ratios, and fixed phase shifts do not enter this DC flow expression.

The public subtype fields are converted to (r_a, x_a) as follows. For a line at common endpoint voltage V, let $Z_{\text{base}} = V^2/S_{\text{base}}$:

$$r_a = \frac{r_{\Omega/\text{km}}\ell}{n_{\parallel} Z_{\text{base}}}, \qquad x_a = \frac{x_{\Omega/\text{km}}\ell}{n_{\parallel} Z_{\text{base}}}. \tag{6}$$

For a transformer, with $z_{\text{elem}} = v_k/100$, $r_{\text{elem}} = v_{kr}/100$, element rating S_a, and $x_{\text{elem}} = \text{copysign}(\sqrt{z_{\text{elem}}^2 - r_{\text{elem}}^2}, z_{\text{elem}})$,

$$r_a = \frac{r_{\text{elem}} S_{\text{base}}}{S_a n_{\parallel}}, \qquad x_a = \frac{x_{\text{elem}} S_{\text{base}}}{S_a n_{\parallel}}. \tag{7}$$

For an impedance element with equal forward and reverse series parameters on element base S_a,

$$r_a = r_{ft}\frac{S_{\text{base}}}{S_a} = r_{tf}\frac{S_{\text{base}}}{S_a}, \qquad x_a = x_{ft}\frac{S_{\text{base}}}{S_a} = x_{tf}\frac{S_{\text{base}}}{S_a}. \tag{8}$$

C.2 Balance, limits, and objective

With $d_i = p_i^d + g_i^s$, where positive p_i^d is withdrawal and negative p_i^d is a fixed injection, the default formulation is

$$\begin{aligned}
\min_{P,\theta} \quad & \sum_{g\in G} \left(c_{2g}P_g^2 + c_{1g}P_g + c_{0g}\right) \\
\text{s.t.} \quad & \sum_{g\in G_i} P_g - d_i = \sum_{a\in A_i^+} P_a(\theta) - \sum_{b\in A_i^-} P_b(\theta), && \forall i \in B, \\
& P_g^{\min} \le P_g \le P_g^{\max}, && \forall g \in G, \\
& -F_a^{\max} \le P_a(\theta) \le F_a^{\max}, && \forall a \in A_F, \\
& \Delta_b^{\min} \le \theta_{i_b} - \theta_{j_b} \le \Delta_b^{\max}, && \forall b \in A_\Delta, \\
& \theta_{i_{\text{ref}}} = 0.
\end{aligned} \tag{9}$$

Here A_F and A_Δ contain branches with finite current thermal ratings and valid angle-difference bounds, respectively. All sets, costs, and limits are determined from the transformed network snapshot. Equations (9) and (5) follow the resistance-aware `DCPPowerModel` convention implemented in PowerModels.jl v0.19.9. The base objective is the convex polynomial

$$f_{\text{DC}}(P,\theta) = \sum_{g\in G}(c_{2g}P_g^2 + c_{1g}P_g + c_{0g}), \qquad c_{2g} \ge 0. \tag{10}$$

In equation 1, this gives a diagonal Q with entries $2c_{2g}$ for generation and zero for angles, c with entries c_{1g} for generation and zero for angles, and $c_0 = \sum_{g\in G} c_{0g}$.

C.3 Instruction-specific hard and soft relations

An instruction-specific hard affine relation uses the constraint residual $h_m(x) = a_m^\top x - b_m$ for $x = (P,\theta)$ and is imposed directly as $h_m(x) \le 0$ or $h_m(x) = 0$. A "greater than" inequality is converted to the "less than" form by reversing the sign of its constraint residual. Let $S_\le$ and $S_=$ index soft inequalities and equalities, respectively, and let $S = S_\le \cup S_=$. For a soft relation k, let $h_k(x)$ denote its affine constraint residual in the same convention. A soft one-sided relation uses one slack $s_k \ge 0$ and imposes $h_k(x) \le s_k$. A soft equality uses the same slack on both sides,

$$h_k(x) \le s_k, \qquad -h_k(x) \le s_k, \qquad s_k \ge 0, \tag{11}$$

and contributes $\lambda_k s_k$ to the objective. A two-sided request is first expanded into two one-sided constraint residuals, each with its own nonnegative slack. When no penalty is stated, the protocol uses $\lambda_k = 5000$. The complete objective is

$$f(P,\theta,s) = f_{\text{DC}}(P,\theta) + \sum_{k\in S} \lambda_k s_k. \tag{12}$$

Table 6: Principal symbols and conventions.

Symbol	Unit	Convention
P_g	MW	Dispatchable active-power output
θ_i	rad	Bus voltage angle
$P_a(\theta)$	MW	Positive in the public source direction
d_i	MW	Fixed active withdrawal $p_i^d + g_i^s$
r_a, x_a	p.u.	Series parameters on the system base
$F_a^{\max}$	MW	Symmetric source-oriented thermal limit
s_k	constraint residual unit	Nonnegative soft-epigraph variable
λ_k	cost per constraint residual unit	Linear soft-violation penalty

D Full-Formulation Synthesis Interface and Benchmark

D.1 Benchmark composition

All 118 tasks begin from independent initial snapshots. Their case and outcome composition is shown in Table 7.

Table 7: Composition of the formal 118-task benchmark.

Case	Optimal	Infeasible	Expected rejection	Total
case30	29	4	7	40
case118	38	1	2	41
case300	35	1	1	37
Total	102	6	10	118

D.2 Fine-grained task-type summary

The task taxonomy is derived directly from each instruction and its reference intent. The taxonomy is multi-label: a task that combines a topology change, a load change, and a soft dispatch requirement contributes to all applicable table rows, so the counts in Table 8 do not sum to 118. The complete instruction-level catalog is included in the accompanying release package.

Table 8: Fine-grained task requirements in the formal benchmark.

Broad type	Fine-grained requirement	Modality	Count
Base formulation	Default hard balance, generator, thermal, and angle constraints without an extra operation or affine constraint	hard	7
Security mode	Thermal limits softened globally or by subtype/scope	soft	7
Network change	Existing branch disconnection/restoration, including invalid, ambiguous, or disconnecting requests	state/rejection	15
Network change	Generator disconnection in the active-power domain	state/rejection	1
Parameter change	System-wide or selected-bus active-load change	state	12
Parameter change	Global, selected, subtype, or all-except thermal-limit change	state	14
Parameter change	Raw physical series-reactance change	state	5
Parameter change	Generator active-power-bound change	state/rejection	10
Parameter change	Generator cost-coefficient change	state	8
Parameter change	Current-output update used by a later constraint	state+soft	3
Generator dispatch	One- or two-sided deviation from current output	hard/soft	18
Generator dispatch	Aggregate generation share relative to total or updated load	mixed	6
Reserve requirement	Remaining available capacity, system-wide or zone-specific	hard/soft	4
Generator coordination	Pairwise, fixed-output, group-floor, online-floor, or ratio relation	hard	10
Interface/flow	Directed interface cap or branch-flow operating band	hard/soft	8
Angle constraint	Explicit reference angle or selected-bus angle band	hard	6
Affine modality	At least one instruction-specific hard affine relation	hard	29
Affine modality	At least one instruction-specific soft affine relation	soft	24
Applicability	Typed rejection required	rejection	10

Table 9: The 10 expected-rejection tasks and their applicability requirements.

Task	Rejection requirement
`case30_A_009`	Nonexistent source branch ID
`case30_A_010`	Synchronous condenser outside the active-power generator domain
`case30_A_026`	Affine request includes a synchronous condenser outside that domain
`case30_A_031`	Voltage-magnitude requirement outside the DC contract
`case30_A_032`	Reactive-power requirement outside the active-power DC contract
`case30_A_033`	Nonlinear product of decision variables
`case30_A_043`	Creation of a new physical branch
`case118_A_030`	Requested outages disconnect the final active network
`case118_A_053`	Endpoint description matches parallel branches non-uniquely
`case300_A_013`	Nonexistent source generator ID

D.3 Interface comparison details

Table 10 details the conditions for the paired interface experiment in Section 3.1. The comparison holds the model, 118 task instructions, generation settings, entry points, mathematical contract, and required return artifact fixed.

Table 10: Software-facing conditions in the paired interface comparison.

Factor	Toolchain-heavy condition	Lightweight public interface
Network carrier	Sanitized DataFrame-backed network object	Ordinary nested Python dictionaries
Missing scalars	DataFrame scalar conventions	Python `None`
Access pattern	Table attributes, rows, and DataFrame operations	Dictionary indexing and iteration
Generic Pyomo help	Minimal import requirement	Non-domain examples of sets, variables, constraints, and objectives
Power-system tool calls	Forbidden	Unavailable and forbidden
Entry points	`modify_network` and `build_model`	Same
Return artifact	Live model and normalized P_g, θ_i, and s_k maps	Same
Identity, units, and semantics	Shared public contract	Shared public contract
Tasks and evaluation	Same 118 instructions and certificates	Same

An independent prompt audit passed all 14 checks, and the 3 network cases had identical carrier-normalized runtime semantic hashes.

Table 11: Paired DeepSeek V4 Flash interface results.

Condition	Build/static fail.	Semantic eval.	Cert. pass	Cert. fail
Toolchain-heavy	15/118	103/118	88/118	30/118
Lightweight	2/118	116/118	101/118	17/118
Difference	-13	$+13$	$+13$	-13

D.4 Inference-time accounting and comparison with the workflow

Tables 2 and 12 sum input tokens (including cached context) and output tokens (including reasoning) from the final normally completed response of each logical generation call across all 118 requests. All rounds are counted regardless of certificate outcome; technical-failure attempts are excluded.

The workflow comparison holds DeepSeek V4 Flash fixed across the 3 full-formulation strategies and the protocolized implementation of the proposed allocation.

Both 3-call strategies improve on single-call synthesis but yield fewer certificate passes than the protocolized workflow, which passes 118/118 tasks. Under this accounting, reflection and the two-builder strategy use 17.97 and 19.42 times as many inference tokens as the protocolized workflow,

Table 12: Certificate pass counts and inference tokens including the protocolized workflow implementing the proposed allocation. Gaps report differences in the number of passed requests relative to the workflow.

Method	Calls/task	Certificate passes	Gap to workflow (tasks)	Inference tokens
Single-call synthesis	1	101/118	−17	7,269,528
Reflection	3	113/118	−5	19,114,560
Two builders + integrator	3	112/118	−6	20,653,542
Protocolized workflow	1	**118/118**	—	**1,063,746**

respectively. For the evaluated model and benchmark, these comparisons provide further empirical support for reallocating modeling responsibilities: the workflow achieves more certificate passes with fewer inference tokens than the 2 tested strategies for increasing inference-time computation.

D.5 Residual failures after additional inference

The following source-reviewed examples come from the final outputs of the 2 strategies using 3 calls each, evaluated with DeepSeek V4 Flash. Each is a formal failure under the same 118-request evaluation used in Section 6.3.

Operations outside the active-power decision domain. In `case30_A_010`, the instruction requests that generators 2 and 5 be taken out of service. Generator 5 is a synchronous condenser with no active-power decision variable; the shared contract requires rejection of the request. Both reflection and the two-builder strategy instead return programs that disconnect both devices and construct a model. Their generated implementations check that the requested identifiers exist but do not check eligibility for the requested operation.

Fabricated electrical parameters for a new line. In `case30_A_043`, the instruction requests a new line between buses 7 and 8 without specifying its electrical parameters. The final integrator program copies the electrical properties of an existing line, assigns a new branch identifier and the requested endpoints, and constructs the model. The expected outcome is rejection; the generated program instead supplies physical network data that the request does not establish.

Generator updates extended to synchronous condensers. In `case300_A_046` (reflection) and `case118_A_042` (two builders plus integrator), the requested parameter changes apply to non-reference generators in the active-power decision domain. Both final programs iterate over all non-reference generator rows, including synchronous condensers. The first incorrectly changes the minimum active-power limit and fixed cost term of 12 synchronous condensers; the second makes the corresponding changes to 35. These state discrepancies show that the generated implementations fail to restrict the update scope to eligible physical devices.

E Protocolized Implementation: JDIMP and JDEF

Section 5 assigns interpretation of the operational meaning of an instruction to the LLM, while deterministic software binds the interpreted requirement to the network instance and constructs the complete DC-OPF. The protocolized implementation organizes the resulting processing through two representations, with compilation and backend processing jointly carrying the deterministic binding and construction responsibilities. JDIMP–JDEF as a whole serves as the interface between the LLM and deterministic software under this separation of responsibilities.

The JSON Dispatch Instruction Message Protocol (JDIMP) is the structured dispatch specification produced by the LLM. It uses set scopes for singleton and batch requests, typed standard-constraint declarations, and instruction-specific continuous affine relations; snapshot-dependent values may remain delayed references. Given JDIMP and the initial snapshot D_0, the deterministic compiler validates and expands the message. It resolves sets and public entity references, evaluates elementwise parameter-update expressions, expands standard and affine specifications, and produces the JSON Dispatch Expanded Form (JDEF) in stable logical entity coordinates.

JDEF is the entity-explicit input to subsequent deterministic processing. The backend applies its topology and parameter operations to obtain D_W, forms the active network, resolves logical coordinates to backend entities and model components, completes the required standard-constraint coverage, and realizes the physical relations, constraints, and objective in the live Pyomo model M_W. Thus, network-instance binding occurs across compilation and backend construction, while JDEF also carries compiled state operations and model specifications. The following subsections detail both representations, give a worked example, and describe their expressive scope and extension points.

E.1 JDIMP STRUCTURE

Table 13: JDIMP structure produced by the LLM.

Level	Fields and allowed structure	Semantics
Root	`protocol`, `meta`, optional `definitions`, `topology`, `parameters`, `model_blueprint`, and `objective`	Complete structured dispatch intent
`meta`	`user_intent`, `is_request_valid`, and optional invalid reason or timestamp	Typed request validity and concise intent
Entity set	Alias; entity in {generator, branch, bus}; strategy in {all, list, all-except}; optional IDs	Set-first scope for singleton and batch requests
Topology update	`connect`/`disconnect` plus a set alias	Existing-entity state change
Parameter update	Set alias, whitelisted attribute, and value expression	Elementwise numeric state update; the old value binds per entity
Standard constraint	Type, hard/soft mode, set, and optional penalty	Balance, thermal, angle, generator-limit, and supported security templates
Affine constraint	ID, optional per-entity expansion, left and right expressions, sense, mode, and optional penalty	Controlled affine customization
Affine expression	Constant plus terms containing coefficients and references	Affine combination only
Value reference	Set variable/attribute or current variable/attribute	References P_g, source-oriented $P_a(\theta)$, θ_i, or whitelisted data
Objective	`minimize_cost`	Convex generation cost plus requested soft penalties

Generator sets contain dispatchable active-power generators, including the primary slack generator where applicable; synchronous condensers and static injections are excluded. Branch sets include lines, transformers, and impedance elements. Reserved role, subtype, and source-zone sets provide common scopes without requiring the LLM to enumerate current entities. Snapshot-dependent values may remain delayed references. Soft standard or affine relations carry an explicit penalty, with 5000 used when the instruction does not supply one. A request outside the affine, active-power contract is represented by the validity fields rather than by an invented protocol construct.

For buses, JDIMP `pd` maps directly to the public-contract field `net.bus.pd_mw`. Both denote the current signed aggregate constant-power demand in MW: a positive value is an active-power withdrawal, whereas a negative value is a fixed active-power injection. The compiler and backend read and update this same current quantity; JDIMP `pd` does not refer to `source_pd_mw`, which retains the original snapshot value.

E.2 JDEF STRUCTURE

Table 14: Entity-explicit JDEF structure.

Level	Fields and allowed structure	Semantics
Root	Protocol, metadata, topology operations, parameter operations, standard specifications, affine constraints, and objective	Deterministic entity-explicit execution form
Topology operation	Action, entity type, and concrete ID	One concrete operation per resolved entity
Parameter operation	Entity, concrete ID, attribute, and numeric new value	Alias and old-value references already resolved
Standard specification	Constraint type, entity, concrete ID, mode, and optional penalty	One explicit standard-constraint target
Expanded affine constraint	Constraint and parent IDs, explicit left/right expressions, sense, mode, penalty, and tags	Per-entity expansion completed
Expanded affine expression	Explicit variable terms, attribute terms, and constant	No set alias or current-entity reference remains
Variable reference	P_g/generator ID, $P_a(\theta)$/branch ID, or θ_i/bus ID	Stable logical entity coordinates
Attribute reference	Concrete entity, ID, attribute, and aggregation	Snapshot values bound to explicit entities
Objective	`minimize_cost`	Same optimization intent as JDIMP

Table 15: Roles and representation in JDIMP and JDEF.

Property	JDIMP	JDEF
Producer	LLM	Deterministic JDIMP-to-JDEF compiler
Consumer	Deterministic compiler	State/model-construction backend
Entity and reference form	Sets, aliases, batch operations, and delayed references	Stable logical entity IDs, entity-explicit operations, and entity-bound attribute references
Expansion	Optional per-entity and aggregation constructs	Fully expanded operations and expressions
Algebra	Controlled affine syntax	Expanded logical expressions; later constraint normalization is unnecessary for protocol meaning

E.3 Worked example: from JDIMP to a Pyomo model

Consider `case30_A_030` from the released task catalog. The instruction first increases the loads at buses 7 and 8 by 10 MW each, then requires the combined output of generators 1 and 2 to be at least 30% of the updated total system load. This example illustrates how entity references are resolved across deterministic compilation and backend construction, as well as the order of state updates and attribute evaluation.

JDIMP: express the requested operations and relation. The message declares `b7_b8` as the bus set $\{7, 8\}$, `g1g2` as the generator set $\{1, 2\}$, and `all_buses` with strategy `all`. The JDIMP message also declares all-generator and all-branch sets for the standard constraints. The instruction-specific fields are shown below; metadata, set declarations, and standard constraints are omitted from this excerpt.

```
{
  "parameters": [
    {"set": "b7_b8", "attr": "pd", "value_expr": "&old + 10"}
  ],
  "model_blueprint": {"affine_constraints": [{
    "id": "g12_ge_updated_load_30pct",
    "lhs": {"terms": [{"coeff": 1, "ref": {
      "kind": "set_var", "set": "g1g2", "var": "Pg"}}]},
    "sense": ">=",
    "rhs": {"terms": [{"coeff": 0.3, "ref": {
      "kind": "set_attr", "set": "all_buses",
      "attr": "pd", "op": "sum"}}]}
  }]}
}
```

The affine relation is hard by default. The complete message also requests hard power balances, generator limits, thermal limits, and angle-difference limits, with objective `minimize_cost`. The total-load expression remains a reference to the `pd` attributes of the bus set.

JDEF: expand the message in logical entity coordinates. The compiler resolves the sets against the case30 snapshot. For the parameter update, `&old` binds separately to each bus's current load: 22.8 MW at bus 7 and 30.0 MW at bus 8. The resulting `parameter_ops` are entity-explicit numeric assignments:

```
{"entity": "bus", "id": 7, "attr": "pd", "new_value": 32.8}
{"entity": "bus", "id": 8, "attr": "pd", "new_value": 40.0}
```

In the expanded affine constraint, the left-hand side contains two `Pg` variable terms, with generator IDs 1 and 2 and coefficient 1 each. The right-hand side contains an `attr_terms` entry with coefficient 0.3, aggregation `sum`, and explicit `pd` references for bus IDs 1 through 30. The standard declarations likewise expand into individual entity targets. At this stage, the load references have concrete identities but retain their attribute form for evaluation after the state changes.

Pyomo: construct the model from the updated state. The backend applies the two parameter operations to obtain D_W, forms the active network, and constructs the standard DC-OPF variables, constraints, and objective. The backend then reads the explicit bus attributes from D_W. The total load has increased from 283.4 MW to 303.4 MW, so the custom relation becomes

$$P_1 + P_2 \geq 0.3 \sum_{i=1}^{30} \mathtt{pd}_i(D_W) = 91.02\,\mathrm{MW}. \tag{13}$$

Here `pd` is the signed active-load attribute; the named total load excludes bus shunt-conductance withdrawals. Mapping the two variable references to `model.pg[1]` and `model.pg[2]` gives the following equivalent Pyomo constraint:

```
model.affine_constraints.add(model.pg[1] + model.pg[2] >= 91.02)
```

The resulting artifact is a live model. The semantic message specifies the requested load update and generation-share relation, while the deterministic implementation supplies the concrete assignments, updated constant, and complete physical formulation.

E.4 Expressive scope and extensibility

The JDIMP–JDEF protocol supports single-period, continuous active-power dispatch with topology changes to existing entities, permitted numeric parameter updates, typed standard constraints, and custom affine relations. The custom expressions combine P_g, source-oriented branch flow $P_a(\theta)$, and bus angle θ_i with numeric coefficients and allowed network attributes. Set summation and per-entity expansion express group generation requirements, signed interface limits, and deviations from current output. Attribute aggregations `sum`, `min`, and `max` evaluate network data to constants during model construction. Hard and soft modes specify exact constraints or penalized violations; the objective is generation cost plus the requested soft penalties.

Requests composed from these operations use the same schema and compilation path. The source-zone aliases and `zone_outward_sum` operator support zone-based entity selection and aggregation: the compiler determines zone membership and branch signs, then emits ordinary entity-explicit JDEF terms consumed by the same model builder.

Adding a decision quantity, physical relation, or entity type requires corresponding definitions and validation in the protocol, deterministic expansion and reference-resolution rules in the compiler, and a mathematical realization in the backend. The affine vocabulary excludes variable–variable products, logical implications, discrete decisions, and AC voltage-magnitude or reactive-power constraints; such requests use the validity and rejection mechanism. The schema also contains a typed `n_minus_1_security` declaration, whose expansion is handled by the backend. N-1 tasks are excluded from the reported 118-task evaluation.

F PUBLIC RUNTIME, SAFETY VALIDATION, AND REPORTS

F.1 PUBLIC RUNTIME ARCHITECTURE

Figure 4 shows the model-construction workflow and its runtime extension.

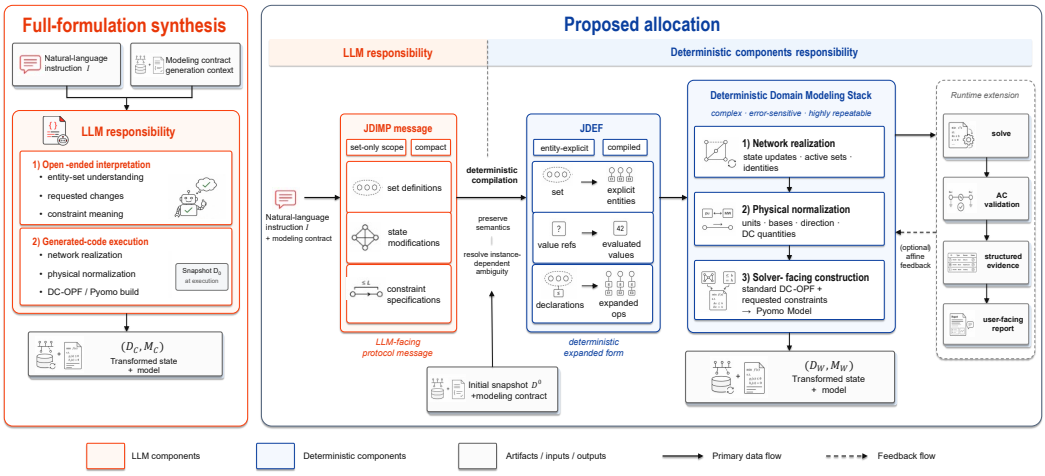


Figure 4: **Full workflow and runtime extension.** JDIMP expresses dispatch intent; deterministic compilation and backend processing use JDEF to bind the request to the instance and construct the model. The runtime adds solution, AC validation, feedback, evidence collection, and reporting. In both modeling settings, numerical snapshots are read during execution.

The LLM's role ends with a schema-valid JDIMP. JDIMP-to-JDEF compilation expands sets and per-entity constructs, resolves public entity references, and evaluates elementwise parameter updates. The backend continues network-instance binding by resolving JDEF's stable logical coordinates to backend entities and active model components while applying state changes and constructing the complete DC-OPF. Solution, validation, feedback, iteration control, and evidence collection are also deterministic under the public contract. Report composition is restricted to connecting and polishing the deterministically collected evidence.

Table 16: Functional components of the public runtime.

Component	Input	Output and responsibility
Interpret instruction	Natural-language instruction	Schema-valid JDIMP
Compile JDIMP to JDEF	JDIMP and current snapshot	Expanded operations, specifications, and references in stable logical coordinates
Apply JDEF and construct model	JDEF and current snapshot	Transformed state and fresh DC-OPF
Solver	DC-OPF model	Dispatch, objective, and dual evidence
Safety validation	DC dispatch and AC network	AC convergence status and physical violations
Structured feedback	Validation findings	DC-expressible parameter corrections or affine cuts
Iteration control	State, model, and feedback	Rebuild, solve, and validate until a terminal condition
Evidence collection	Final solve and validation records	Deterministic structured evidence payload
Report composition	Structured evidence payload	Evidence-grounded reader-facing report

F.2 SAFETY-VALIDATION AND REPORTING EXAMPLES

Safety validation injects the DC active-power dispatch into an AC power-flow calculation and evaluates the resulting AC state. Evidence-based reporting is grounded by a deterministic evidence

collector, which extracts the relevant solve, validation, and iteration records and inserts them into a structured report payload. The LLM is used only to connect the collected evidence items into a coherent account and polish the presentation; the LLM does not generate the underlying measurements or validation decisions. Depending on the run outcome and the available evidence, a report may cover the following content.

Table 17: Potential contents of an evidence-based execution report.

Evidence group	Deterministic source	Possible report content
Execution outcome	DC solve, iteration, and AC-PF termination records	DC status, terminal iteration, AC-PF convergence, and validation outcome
Dispatch and cost	Final DC solution and available AC-PF result	Generator dispatch, DC objective and penalty, AC-scenario generation, and slack adjustment and cost
Physical validation	AC-PF state and deterministic limit checks	Branch loading, bus voltage, active- and reactive-power violations, and physical-feasibility status
Economic diagnostics	DC primal and dual records	Binding limits, thermal marginal values, and nodal marginal values
Feedback history	Per-iteration validation and feedback records	Triggering violations, deterministic corrections or affine cuts, and the validation trajectory
Evidence availability	Collector field-presence and termination records	Explicitly unavailable or indeterminate quantities when AC-PF evidence is not produced

Figure 5 illustrates structured feedback for the case118 base-dispatch instruction. It shows how an AC overload signal is converted into a directional DC feedback constraint and how the maximum AC loading evolves across iterations.

F.2.1 CASE118 EVIDENCE-BASED REPORT

In the flat-case setting, the DC-OPF is optimal after an 8-iteration DC–AC loop. The AC power flow converges and passes the primary validation criterion, while 8 reactive-power violations remain; accordingly, the AC result is not fully physically feasible.

Table 18: case118 execution and validation summary.

Reported quantity	Value
DC status / iterations	Optimal / 8
DC generation cost / penalty	93,599.73 / 0.00 cost units
Total DC / AC generation	4,242.00 / 4,419.59 MW
AC-scenario generation cost	98,174.17 cost units
Additional slack generation / cost	177.59 MW / 4,574.44 cost units
AC power flow / primary validation	Converged / passed
Remaining constraint violations	0 P, 8 Q, 0 branch, 0 voltage
Fully physically feasible	No

The highest-loaded standard branch in the final DC model is line 130 (buses 88–91), at 185.33 MW against a 186.00 MW limit; all standard thermal dual values are 0. The leading reported nodal marginal value is 25.76 cost/MW at bus 58. The 2 nonbinding directional cuts remain as protective surfaces because AC validation detected overload risk during the DC–AC loop: line 98 was triggered by 100.76% loading and ends at 98.17%, while line 152 was triggered by 100.91% and ends at 99.92%. The report distinguishes these triggering observations from the final AC state.

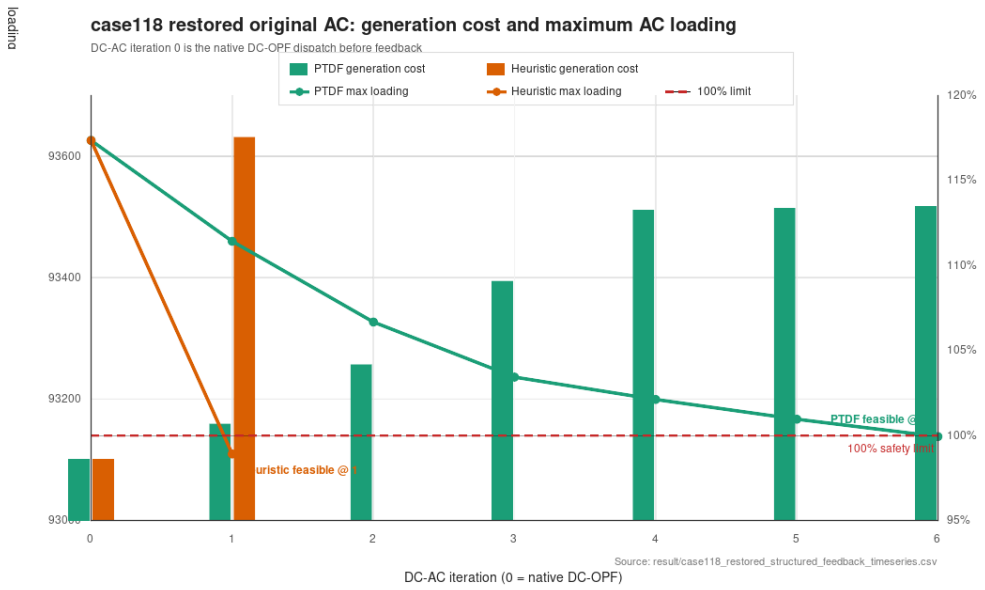


Figure 5: Structured feedback for the case118 base economic dispatch under the restored-original-AC variant. Iteration 0 is the native DC dispatch. PTDF feedback brings maximum AC loading below 100% at zero-based iteration 6; the heuristic comparison reaches that threshold at iteration 1. This figure and the report summarized below concern the same base instruction but distinct AC-data variants and are not treated as one run.

Table 19: case118 final DC dispatch and corresponding AC result.

Generator	Bus	P^{DC} (MW)	P^{AC} (MW)	Q^{AC} (MVAr)	V (p.u.)
5	9	505.0000	505.0000	-81.0294	1.0000
6	11	-0.0000	-0.0000	85.6377	1.0000
11	24	117.0844	117.0844	-142.6518	1.0000
12	25	485.0000	485.0000	105.2366	1.0000
14	30	17.0000	17.0000	36.6429	1.0000
20	45	20.0000	20.0000	21.3426	1.0000
21	48	223.0000	223.0000	31.4344	1.0000
22	53	53.0000	53.0000	53.9506	1.0000
25	58	308.0000	308.0000	26.6109	1.0000
26	60	195.0000	195.0000	-45.6138	1.0000
28	64	-0.0000	-0.0000	256.7195	1.0000
29	65	-0.0000	-0.0000	-188.9877	1.0000
37	79	509.0000	509.0000	-32.5374	1.0000
39	86	-0.0000	-0.0000	2.5523	1.0000
40	88	637.0000	637.0000	-94.3713	1.0000
45	99	653.0000	653.0000	-56.6033	1.0000
46	102	39.7811	39.7811	24.0225	1.0000
51	110	-0.0000	-0.0000	-1.0000	1.0000
30	68	480.1345	657.7243	-234.1274	1.0000

Table 20: Remaining case118 reactive-power violations after AC validation.

Generator	Bus	$Q_{\min}$	$Q_{\max}$	Q^{AC}	Violation (MVAr)
6	11	-35.0000	43.0000	85.6377	42.6377
11	24	-47.0000	111.0000	-142.6518	95.6518
14	30	-9.0000	9.0000	36.6429	27.6429
20	45	-10.0000	10.0000	21.3426	11.3426
22	53	-27.0000	27.0000	53.9506	26.9506
28	64	-67.0000	200.0000	256.7195	56.7195
29	65	-67.0000	200.0000	-188.9877	121.9877
45	99	-50.0000	155.0000	-56.6033	6.6033

F.2.2 CASE300 EVIDENCE-BASED REPORT

The case300 DC-OPF is optimal with generation cost 517,851.0752 and a penalty cost of 0. The subsequent AC power flow does not converge. Consequently, AC feasibility is indeterminate: the report provides neither an AC generation cost nor AC violation counts and does not convert nonconvergence into a claim of physical infeasibility.

Table 21: case300 execution and validation summary.

Reported quantity	Value
DC status / iterations	Optimal / 1
DC generation cost / penalty	517,851.0752 / 0.0 cost units
AC power flow	Did not converge
AC physical feasibility	Indeterminate
AC cost, slack adjustment, violation counts	Unavailable

Table 22: Complete case300 DC dispatch reported before AC nonconvergence.

Generator	Bus	P^{DC} (MW)	Generator	Bus	P^{DC} (MW)
6	83	852.0000	39	232	341.0000
7	90	-0.0000	40	235	881.0000
8	91	-0.0000	41	237	-0.0000
9	97	93.0000	42	238	-0.0000
10	107	2.7648	43	240	954.7012
11	118	1800.4686	44	241	-0.0000
12	123	321.0000	45	242	342.0000
15	140	297.0000	46	7000	-0.0000
16	142	248.0662	47	7001	670.0000
17	145	91.0000	48	7002	1401.0000
18	146	-0.0000	49	7010	668.0000
19	148	-0.0000	50	7011	626.0000
20	151	182.2621	51	7016	490.0000
21	152	242.0000	52	7022	-0.0000
23	169	119.7677	53	7023	543.0000
25	175	-0.0000	54	7038	-0.0000
26	176	125.0000	55	7043	132.0000
27	184	250.0000	57	7054	150.0000
28	185	2465.0000	58	7056	-0.0000
29	186	1624.0000	59	7060	1175.3350
30	189	-0.0000	60	7061	483.4480
31	190	1551.3363	61	7070	203.0000
32	197	430.0000	62	7129	1520.0000
33	212	-0.0000	63	7138	-0.0000
34	219	460.0000	64	7165	-0.0000
35	220	554.0000	68	9053	84.0000
36	221	-0.0000	69	9054	-0.0000
37	226	436.0000	56	7048	718.0000
38	229	-0.0000			

From the DC perspective, line 153 (buses 118–120) is binding at 504.0 MW and has the largest reported thermal marginal magnitude, 115.59 cost/MW. The reported nodal marginal value at bus 120 is 77.55 cost/MW. Generator 44 is at its lower bound of 0 with a 77.87 cost/MW mathematical marginal value; allowing negative output is not an actionable operating option. These DC quantities remain available even though no usable AC validation result is produced.

G PROTOCOL COVERAGE

We assess the JDIMP–JDEF protocol's coverage of modeling requirements within this study's stated scope of single-period, continuous DC-OPF. The assessment examines 9 independently authored requests on PGLib-OPF v23.07 `case1354_pegase`, with 1,354 buses, 260 active-power generators, and 1,991 branches.

G.1 REQUEST AUTHORING WITH PROTOCOL DETAILS WITHHELD

The independent authoring LLM was `gpt-6-astra`. In a fresh context, it received a domain-scope description and raw network facts. The scope covered single-period, continuous DC economic dispatch: specified equipment-status changes, numerical updates to existing data, standard operating constraints, additional finite affine relations, and linear penalties for soft violations. The supplied domain scope excluded new equipment, multiperiod states, discrete decisions, and AC-only requirements. The author received no protocol syntax, schemas, implementation, benchmark tasks, encoding examples, or solution feedback, and was instructed to access only the supplied packet.

The packet was extracted directly from the original MATPOWER tables, using a fixed random seed to sample 24 generators, 16 buses, and 24 branches, together with zone summaries and 4 deterministically selected examples of parallel branches. Sampling used no optimization results. The author chose the task mix and supplied 9 English requests with intended semantics. We fixed the original output before the end-to-end evaluation and retained all 9, including the request outside the unchanged protocol and the infeasible request. Network data, author inputs and outputs, run configurations, saved model responses, and per-request certificate checks were retained as study artifacts.

Independence was operationalized through a fresh authoring context and withheld protocol details; a protocol-aware coordinator prepared the domain-scope packet.

G.2 END-TO-END WORKFLOW EVALUATION

We evaluated the complete workflow on the 8 requests that are expressible under the unchanged protocol. Request 04 was excluded before generation because its angle-bound update is outside the parameter-access boundary described below. For each included request, DeepSeek V4 Flash received the fixed workflow prompt with 4 in-context examples and that request's English text, but no instance-specific network values. Generation used thinking enabled, high reasoning effort and no tools or temperature setting. Each saved model response was processed by the production JSON extraction and JDIMP validation path, followed by deterministic expansion, network-state realization, Pyomo model construction, and solution on a fresh network instance. Table 23 reports the resulting per-request outcomes; all requests retain the existing generation-cost objective except for the specified cost update or added violation penalty.

Table 23: 9 independently authored requests on `case1354_pegase`. IDs abbreviate `X1354-01`–`X1354-09`. A pass denotes successful model generation followed by a complete certificate pass. The separately marked manual-write check for request 04 does not count as end-to-end workflow execution.

ID	Requested operations and relations	Workflow outcome
01	Increase the fixed injection at bus 2782 by 25% and demand at bus 6891 by 20%; derate generator 217 to a 650 MW maximum limit.	Pass; optimal
02	Take generator 41 offline; retain at least 15,000 MW of total upward headroom across remaining online non-slack generators.	Pass; optimal
03	Take parallel branch 899 offline, put 900 in service, and set branch 900's symmetric thermal limit to 250 MW.	Pass; optimal
04	Increase branch 19's raw physical series reactance by 20%; replace its source-oriented angle bounds with -5° and $+8^\circ$.	Extension required; manual-write check passed
05	Increase generators 1, 47, and 74 by 300 MW in total from current values; require nonnegative increments, at least 80 MW from generator 1, and a 2:1 increment ratio for 47 and 74.	Pass; optimal
06	Bound the sum of flow magnitudes on branches 75 and 128 by 600 MW and require at least 100 MW on branch 128 in its source direction.	Pass; infeasible
07	Soften branch 1609's 300 MW thermal threshold, charging 1,000 cost units per MW of excess; retain a 330 MW absolute hard ceiling.	Pass; optimal
08	Halve generator 217's linear cost; cap the combined signed dispatch of generators 1, 47, 74, and 198 at 12% of total online zone-5 dispatch, including the reference generator.	Pass; optimal
09	Set generator 64's minimum limit to -80 MW; require generators 64 and 75 to absorb 120 MW in total, with at least 30 and 50 MW respectively.	Pass; optimal

All 8 model outputs compiled as valid JDIMP and passed the complete certificate. Seven requests were solved to optimality, and request 06 was correctly found infeasible. In the generated specification for request 06, the magnitude budget was expressed through the following finite set of affine constraints. For source-oriented branch flows,

$$|P_{75}(\theta)| + |P_{128}(\theta)| \le 600 \quad \Longleftrightarrow \quad \sigma_1 P_{75}(\theta) + \sigma_2 P_{128}(\theta) \le 600 \text{ for all } (\sigma_1, \sigma_2) \in \{-1, 1\}^2. \tag{14}$$

These four affine constraints, together with $P_{128}(\theta) \ge 100$, express the request without additional decision variables. Feasibility is a property of the resulting dispatch problem, separate from whether the instruction can be represented.

For certificate evaluation, a separate implementation constructed a sparse linear reference model directly from the raw MATPOWER tables and each request's intended semantics, using the convention in Appendix C. The reference implementation used no production loader, protocol compiler, or shared DC expression helper. The comparison covered 21,810 specified network-state values per request, all constraints and variable bounds, and objective coefficients in a common source-identity coordinate system. Constraints were compared in both directions after normalization, allowing equivalent scaling, duplicate constraints, and satisfied constant constraints.

Across the 8 end-to-end runs, the state comparison found no mismatches, all variable sets matched, and no hard-row mismatches were found. The largest normalized row-coefficient difference was 3.34×10^{-16}, objective coefficients agreed exactly, and the largest absolute objective difference for the 7 optimal cases was 1.73×10^{-8}, below the 10^{-4} comparison tolerance. The production workflow and reference implementation used the HiGHS solver family through Pyomo and SciPy, respectively; their model constructions are independent and share solver technology. Structural checks also cover requirements that are inactive at the optimum.

G.3 PARAMETER-ACCESS BOUNDARY

Request 04 combines two changes within the broad parameter-change category. The raw-reactance update is supported. Replacement of stored branch angle-difference bounds is an *unsupported parameter-access subtype*: these bounds are present in the public network and enforced by the backend, but are not currently exposed for modification through the JDIMP parameter whitelist. Adding affine angle inequalities can represent the requested operating envelope, but the instruction explicitly requires replacing the stored bounds. Additional affine inequalities alone do not perform the required state transformation.

For the angle-bound replacement in request 04, we separately wrote the requested bounds to the existing public fields and ran the unchanged reactance-update and model-construction path. State, constraints, and objective matched the raw-data reconstruction, yielding an optimal solution. This parameter-write check establishes backend support for the requested data. Supporting this subtype through JDIMP would require exposing the lower and upper angle-bound attributes with units, source orientation, compiler reads, state writes, validation, and versioned documentation. It requires no new decision quantity, physical relation, or solver capability.

G.4 COVERAGE OF MODELING REQUIREMENTS

All 9 independently authored requests fall within requirement types already represented in the formal benchmark (Table 24). These correspondences relate the protocol coverage checks to the benchmark's modeling requirements. The requests combine operations and relations with different entity selections and parameter settings. Request 04 identifies an unsupported parameter-access subtype within the existing parameter-change category.

Table 24: Requirement-type correspondences between the independently authored requests and formal benchmark tasks. IDs refer to Table 23; the complete benchmark instructions are included in the accompanying release package. Correspondences concern shared operations and relations; the full combinations and parameter settings differ.

Requirement type	IDs	Corresponding components	Benchmark examples
Parameter change	01, 03, 04, 08, 09	Signed-demand, thermal-limit, reactance, generator-bound, and cost updates. Stored angle-bound replacement (04) is an unsupported parameter-access subtype.	`case30_A_022`, `case30_A_025`, `case118_A_011`, `case300_A_007`, `case118_A_009`
Network status	02, 03	Generator-status requests and branch outage/restoration. The listed generator-status task is an expected rejection.	`case30_A_010`, `case118_A_014`, `case300_A_016`
Reserve requirement	02	Aggregate remaining capacity of non-slack generators.	`case118_A_048`
Generator coordination	05, 09	Output deviations, group sums, fixed-output relations, and ratios.	`case30_A_014`, `case118_A_051`, `case118_A_054`, `case300_A_054`
Flow constraints	06	Signed interface sums and affine flow bounds; the magnitude budget is composed as in Eq. (14).	`case30_A_016`, `case30_A_029`
Hard/soft limits	07	Soft flow thresholds alongside hard thermal ceilings.	`case30_A_028`, `case300_A_025`
Generation share and grouping	08	Aggregate generation shares and zone-based entity aggregation.	`case118_A_025`, `case300_A_052`

The protocol's expressive scope is defined by the permitted entity types, state operations, decision quantities, and relation types. The 8 successful end-to-end runs cover combinations of requirements within this scope, and the independent checks verify their network-state realization and model construction.

H RESPONSIBILITY-ALLOCATION AND WORKFLOW COMPARISONS

The intermediate-allocation comparison in Section 6.4 evaluates the primitive helper and the constraint helper with DeepSeek V4 Flash and GPT 5.6 Luna. Each reported model–helper condition contains one scored output for each of the same 118 English task messages used in the main comparison, comprising 102 optimal, 6 infeasible, and 10 expected-rejection tasks. This gives 472 reported model–task outputs. Both conditions use a single model call per task, high reasoning effort, no task examples, and no tools. Their system prompts are deterministic variants of the frozen full-formulation prompt, with the same public network-data contract and user messages. Technical recovery follows the common generation policy; scoring uses the first accepted normal response for each task and retains its semantic and code errors. All outputs are assessed by the same certificate evaluator as the main comparison.

For the primitive helper, the final result retains 117 outputs from the original complete run and unconditionally replaces `case300_A_028` for both models with fresh outputs generated from the corrected `zone 3` message. The replacement was applied regardless of whether the new output passed, and the superseded `area 3` outputs are excluded from the reported scores.

At generation time, the model receives the instruction, public data contract, and the condition-specific helper documentation. Its generated Python code later receives the public network snapshot and performs entity resolution, scope selection, applicability checks, and topology checks at execution. The helpers are task-independent: their interfaces contain no task-specific entity selection or request interpretation.

H.1 PRIMITIVE HELPER

This condition retains the full-formulation entry points `modify_network(net)` and `build_model(modified_net)`. The first updates the public network dictionary; the second returns a live Pyomo model together with the `Pg`, `Theta`, and `Soft` maps. Candidate code may additionally call a task-independent library that performs the following local transformations on explicitly identified entities:

- `branch_rx_on_system_base` converts the current native fields of one branch into its numeric system-base resistance and reactance, while `flow_coefficient` returns the corresponding resistance-aware scalar coefficient;
- `source_alignment_sign` validates the branch endpoints and returns the sign relating the current subtype endpoint order to the public source direction;
- `get_raw_reactance` and `set_raw_reactance` read or update one branch's raw physical series reactance through subtype-specific native-field conversion;
- `fixed_active_withdrawal_mw` returns the fixed active-power withdrawal of one bus; and
- `generator_polynomial_cost_term` evaluates one generator's current polynomial cost coefficients at a caller-supplied scalar.

The library does not construct a branch-flow expression, nodal balance, active set, incidence relation, cross-entity aggregation, constraint, or objective. Candidate code selects the active buses and generators, combines the local quantities into every MW flow and nodal-balance expression, aggregates generator cost terms, and constructs every variable, bound, standard and instruction-specific constraint, soft epigraph, objective, and required return map. Thus, deterministic software performs only task-independent local transformations; equation assembly and complete Pyomo model construction remain responsibilities of the generated code.

H.2 CONSTRAINT HELPER

The constraint helper defines `configure(net, h)` as its single entry point, replacing the 2 full-formulation functions. The network argument is the initial public snapshot, and candidate code records its requested operations through a task-independent command interface. `h.get` and `h.set` read and write whitelisted bus, generator, and branch state through explicit public identifiers.

`h.pg`, `h.theta`, and `h.flow` create symbolic references to generator dispatch, bus angle, and source-oriented branch flow. Candidate code combines these references into affine expressions. It uses `h.add` to declare hard or soft instruction-specific rows and `h.standard` to select hard or soft treatment for generator, thermal, or angle-difference limits. Each soft declaration includes its requested penalty or the public default.

After `configure` returns, deterministic software applies the recorded state changes and constructs the complete live Pyomo model. This construction instantiates the active variable domains, physical branch-flow relations, nodal balances, generator bounds, standard thermal and angle-difference constraints, reference-angle fixing, declared affine rows, soft epigraphs, and the complete objective. Candidate code remains responsible for interpreting the request, resolving every referenced entity and scope against the snapshot, choosing the state operations and constraint semantics, and issuing the corresponding declarations with explicit public identifiers.

Table 25: Responsibility boundaries in the 2 intermediate helper conditions.

Responsibility	Primitive helper	Constraint helper
Interpret applicability and operational meaning	Candidate code	Candidate code
Resolve entities and scopes	Candidate code over the public snapshot	Candidate code over the public snapshot
Apply state changes	Candidate code edits the public dictionary; the library converts raw reactance	Candidate code declares explicit `get`/`set` operations; the backend applies them
Convert per-entity physical data	Library converts native branch parameters, direction, raw reactance, fixed withdrawal, and one-generator cost terms	Backend performs these conversions during model construction
Assemble flow, balance, and total-cost expressions	Candidate code combines local helper outputs across selected entities	Backend constructs the complete default physical model
Instantiate variables and standard constraints	Candidate code	Backend
Add instruction-specific affine relations	Candidate code constructs Pyomo constraints	Candidate code declares affine rows; the backend instantiates them
Implement soft constraints and penalties	Candidate code constructs slacks, epigraphs, and objective terms	Candidate code declares mode and penalty; the backend constructs slacks, epigraphs, and objective terms
Produced artifact	Candidate-built live Pyomo model and variable maps	Recorded commands compiled by the backend into the live Pyomo model and variable maps

The primitive helper transfers only task-independent transformations on explicitly identified entities; generated code retains active-set selection, cross-entity aggregation, physical-relation assembly, and complete model construction. The constraint helper additionally transfers variables, standard constraints, soft constraints, and objective construction to deterministic software, while generated code continues to interpret the instruction and bind it to explicit network entities.

H.3 RESIDUAL FAILURES UNDER INTERMEDIATE RESPONSIBILITY ALLOCATIONS

Table 26 classifies all 66 failures under the 2 intermediate helper conditions using the four categories and first-error attribution in Section L.3. Each condition comprises 118 tasks per model, or 236 model–task outputs. Aggregate rows pool these outputs; the model rows retain differences in error composition.

Table 26: First-error categories for helper failures on 118 tasks per model. Each failure is counted once. Counts describe the first confirmed error; later errors in an interrupted candidate may remain unobserved.

Helper allocation	Model	Physical relations	Entity binding and scope	Constraint semantics	Program implementation	Total
Primitive helper	DeepSeek V4 Flash	2	5	3	6	16
Primitive helper	GPT 5.6 Luna	8	13	0	6	27
Primitive helper	Aggregate	10	18	3	12	43
Constraint helper	DeepSeek V4 Flash	0	8	0	0	8
Constraint helper	GPT 5.6 Luna	0	14	0	1	15
Constraint helper	Aggregate	0	22	0	1	23

Primitive helper. DeepSeek V4 Flash's 16 failures comprise 2 physical-relation errors, 5 entity-binding and scope errors, 3 constraint-semantics errors, and 6 program implementation errors. The physical errors reverse branch incidence in nodal balances or apply angle limits to a flow expression. The entity errors concern the active-generator domain, ambiguity among parallel branches, and public entity scope. All 3 semantic errors write a dispatch condition into public network state. The program errors include invalid Pyomo names, undefined variables, invalid imports, and incorrect initialization or control flow.

GPT 5.6 Luna's 27 failures comprise 8 physical-relation errors, 13 entity-binding and scope errors, and 6 program implementation errors. The 8 physical failures arise when candidate code combines correct local coefficients and direction information into incorrect angle differences, flow directions, or bus-incidence signs; they are not errors in the helper's per-entity conversions. The entity errors include confusion between public identifiers and backend row keys, incorrect inclusion or omission of slack generators and synchronous condensers, and erroneous applicability checks. The program errors comprise syntax, name, record-access, unpacking, and control-flow failures.

Constraint helper. The 22 entity/scope failures comprise 14 identifier/row-key errors (2 Flash, 12 Luna), 7 synchronous-condenser scope errors (5 Flash, 2 Luna), and 1 omission of negative-net-load updates (Flash). The latter is an update-scope error: the candidate scales only positive loads although the request covers all load magnitudes. Luna's remaining failure accesses an unsupported helper field. Of its 12 identifier errors, 11 repeat the same false-duplicate check: the candidate tests public bus identifiers against a dictionary keyed by table rows and incorrectly rejects unique identifiers. Helper validation correctly rejects invalid candidate operations, including out-of-domain generator updates; these remain failures to implement valid user requests.

Interpretation of the comparison. For these 2 models, full-formulation synthesis has 82 failures, the primitive helper has 43, and the constraint helper has 23. The combined first-error count in physical relations, constraint semantics, and program implementation decreases from 65 to 25 and then to 1 as more construction responsibility moves to deterministic software. Entity-binding and scope errors remain at 17, 18, and 22, respectively, because both helper interfaces leave instance-dependent entity and scope resolution to generated code.

The comparison describes independently generated candidates under different interfaces and prompts. Its error distributions show that deterministic local transformations reduce failures associated with low-level physical data without removing the LLM's responsibility for assembling equations or binding requirements to the network instance. The constraint helper removes nearly all remaining construction errors, while leaving entity-binding and scope as the dominant residual category. Together with the proposed-allocation results, this pattern supports transferring both responsibilities to deterministic software.

H.4 WORKFLOW FOUR-SHOT VERSUS ZERO-SHOT

The four in-context examples were enumerated while JDIMP was being designed, before the benchmark task set was designed. They cover standard economic dispatch; two instruction-specific affine constraints, comprising a ramp constraint and a relation between aggregate generation and aggregate load; and a compound instruction that disconnects a branch and adjusts branch capacities. They were therefore not selected from the 118 benchmark requests.

This controlled comparison uses DeepSeek V4 Flash, the same 118 user messages, the same generation settings, and the same end-to-end certificate procedure. The zero-shot condition removes only the 4 user/output examples from the public workflow prompt; all remaining prompt content and task definitions are fixed.

Table 27: Workflow four-shot versus zero-shot certificate outcomes.

Condition	Task definition	Total	Pass	Fail
Four-shot	Full public workflow prompt with 4 user/output examples	118	118	0
Zero-shot	Identical prompt and tasks with only those examples removed	118	115	3

H.5 REPEATED FULL-RUN EVALUATION

To examine variation across independent generations, we repeated both evaluated responsibility allocations twice for GPT-5.6-Luna and Gemini-3.6-Flash. Together with the run reported in the main comparison, this gives 3 complete runs for every model–allocation pair. Each additional run independently regenerated all 118 outputs; candidate outputs were not reused across runs, ordinary semantic failures were not retried, and no output repair was applied. All runs use the same 118-task corpus, rejection profile, task messages, and certificate evaluator. A pass is a `PASS_FULL`, `PASS_REJECTION`, or `OUTCOME_MATCH_INFEASIBLE` decision.

Table 28: Certificate passes in 3 complete runs per model and responsibility allocation. Run 1 is the run reported in the main comparison; Runs 2 and 3 are independent repetitions. The paired gain is the proposed-allocation pass count minus the full-formulation synthesis pass count in the same run.

Model	Run	Full-formulation synthesis	Proposed allocation	Paired gain
GPT-5.6-Luna	1	53/118	117/118	+64
GPT-5.6-Luna	2	52/118	116/118	+64
GPT-5.6-Luna	3	53/118	117/118	+64
Gemini-3.6-Flash	1	78/118	117/118	+39
Gemini-3.6-Flash	2	83/118	117/118	+34
Gemini-3.6-Flash	3	85/118	116/118	+31

For GPT-5.6-Luna, full-formulation synthesis obtains 52–53 passes, with a mean of 52.67/118 (44.63%) and a sample standard deviation of 0.49 percentage points. The proposed allocation obtains 116–117 passes, with a mean of 116.67/118 (98.87%) and a sample standard deviation of 0.49 percentage points. For Gemini-3.6-Flash, the corresponding full-formulation synthesis range is 78–85 passes, with a mean of 82.00/118 (69.49%) and a sample standard deviation of 3.06 percentage points; the proposed allocation again obtains 116–117 passes, with a mean of 116.67/118 (98.87%) and a sample standard deviation of 0.49 percentage points. The within-run pass-count gain is positive in all 6 paired comparisons: it is 64 passes in every GPT-5.6-Luna run and 31–39 passes across the Gemini-3.6-Flash runs.

At the task level, the proposed allocation passes all 3 runs on 115 of 118 tasks for GPT-5.6-Luna and 116 of 118 tasks for Gemini-3.6-Flash. No task fails under the proposed allocation in all 3 runs for either model. Thus, the few workflow failures vary across generations rather than defining a task that fails consistently in these repetitions. These experiments characterize run-to-run variation for the two selected models; the 9-model main comparison continues to report one run per model.

I Instance-Level Formulation Equivalence and Full-Formulation Fidelity

The synthesis task requires faithful implementation of the instruction and public modeling contract. The automated evaluator compares the mathematical objects induced on a frozen instance. This appendix defines these two notions of correctness and examines situations in which an implementation violation leaves the evaluated objects or checked task outcome unchanged.

I.1 Two notions of correctness

Instance-level formulation equivalence. For a fixed instruction I and initial snapshot D_0, let p_I denote the generated candidate program, with $(D_C, M_C) = p_I(D_0)$. Here D_C, M_C are the returned state and model, and D_R, M_R are their reference counterparts prescribed by the instruction and modeling contract. For a feasible task, a candidate is *instance-equivalent* to the reference when it produces the same transformed state, the same objective polynomial and associated soft-constraint formulation after semantic alignment, and the same hard feasible region in public coordinates. Writing $f_X(x, s)$ for the complete objective in equation 1, including soft penalties, this requires

$$D_C = D_R, \qquad f_C(x, s) \equiv f_R(x, s), \qquad \mathcal{F}_C = \mathcal{F}_R, \tag{15}$$

together with agreement of the associated soft-constraint formulation. Here $\equiv$ denotes coefficient-wise polynomial identity after alignment of the public variables and soft slacks within the prescribed soft-constraint representation (Appendix J.2). $\mathcal{F}_X$ admits joint assignments to $x = (P, \theta)$, as in Section 4.1. Matching only an optimal dispatch or optimal value is insufficient. The evaluator applies the numerical tolerances in Appendix J.3. Infeasible tasks retain the state and objective checks together with solver-confirmed infeasibility; expected-rejection tasks must raise the prescribed exception.

Full-formulation fidelity. A candidate is *faithful* when its implementation realizes the entity scopes, state-transformation rules, and physical and operational relations prescribed by the instruction and public modeling contract, and its task outcome satisfies the prescribed behavior. Modelable tasks are assessed through the returned state and live formulation; expected-rejection tasks are assessed against the prescribed rejection behavior. A violation remains a fidelity violation when its effects are masked on the frozen instance. Equivalent algebraic representations are assessed against the same requirements; fidelity does not require identical program syntax, Pyomo names, or constraint ordering. For the workflow, assessment includes deterministic network-instance binding, construction, and the expected completion of standard constraints.

Expected-rejection tasks. For expected-rejection tasks, the automated evaluator checks for the prescribed exception type, `UnsupportedRequest`; this check does not establish that the program rejected for the prescribed reason. Automating comparison of declared rejection reasons would require semantic interpretation of exception explanations or a predefined set of reason-specific exceptions or codes. Publishing such a set in the public contract could also provide additional cues about the expected rejection conditions. Review can inspect the exception-raising conditions and control flow to determine whether the rejection follows the prescribed applicability rule.

Instance equivalence alone therefore does not establish fidelity: distinct operation scopes or constraint implementations can induce the same mathematical objects on the evaluated instance. Sections I.3 and I.4 illustrate two classes of errors that can preserve instance-level formulation equivalence while violating full-formulation fidelity: constraint errors that leave the feasible region unchanged, and state-update errors masked by zero-valued fields. Section I.5 quantifies the observed distinction through a complete DeepSeek V4 Flash output review and a targeted review of zero-valued quadratic-cost update tasks. The benchmark statistics and these reviews jointly support the paper's conclusions that the evaluated LLMs do not reliably perform end-to-end physical modeling in full-formulation DC-OPF synthesis, and that assigning interpretation of the operational meaning to the LLM while assigning network-instance binding and complete DC-OPF construction to deterministic software improves reliability.

I.2 Choice of the automated correctness criterion

Instance-level formulation equivalence. This criterion gives the automated evaluation explicit mathematical comparison objects: the transformed state, objective polynomial and soft-constraint formulation, and hard feasible region in public coordinates. Coefficients can be compared directly in these aligned coordinates, and affine hard feasible regions admit the bidirectional containment criterion in Section 4.2. A decisive containment failure supplies a separating witness. The criterion permits different names, orderings, and algebraic representations while specifying exactly which properties must agree on the evaluated instance.

Perturbation tests. Tests on valid modified networks can expose additional errors whose effects are masked on the original instance. However, such tests require additional assumptions about which fields may change, which entity identities and validity conditions remain fixed, how the instruction is interpreted on the modified snapshot, and how the reference state and formulation are reconstructed. They therefore provide only an approximate assessment of full-formulation fidelity within the adopted assumptions and tested conditions.

Review. Direct inspection of implementation code assesses whether a candidate faithfully constructs the prescribed state and model. The reviews in Section I.5 inspect candidate source and deterministic workflow construction, revealing fidelity violations even when their numerical effects are masked on the frozen instance. However, review is costly and difficult to scale.

I.3 Constraint errors that preserve the feasible region

Constraint errors can leave the state and objective unchanged while preserving the hard feasible region. Such errors have zero effect on the mathematical objects compared on the frozen instance, whether the region is nonempty or empty.

Omission of an implied prescribed constraint. Let $\mathcal{F}_{-m}$ denote the hard feasible region after removing a prescribed inequality $h_m(x) \le 0$. If

$$\mathcal{F}_{-m} \subseteq \{x : h_m(x) \le 0\}, \tag{16}$$

then adding this inequality leaves $\mathcal{F}_{-m}$ unchanged. A feasible-region comparison consequently has no separating witness for its omission. This is implication over the entire remaining feasible region; slack at an optimal point alone would not establish it.

The default branch angle-difference bounds on the frozen case30 provide a concrete example. All 41 active branches have hard thermal limits and angle-difference bounds of $\pm 30^\circ$. Under the declared flow convention in equation 5, each thermal limit implies

$$|\theta_{i_a} - \theta_{j_a}| \le \frac{F_a^{\max}}{S_{\text{base}}|\beta_a|} = \frac{F_a^{\max}(r_a^2 + x_a^2)}{S_{\text{base}}|x_a|}. \tag{17}$$

Across the 41 branches, the largest of these upper bounds is 0.30831472 rad, approximately 17.665132°, which is strictly below 30°. It occurs on source branch 40, from bus 8 to bus 28, with $r_a = 0.0636$ p.u., $x_a = 0.2$ p.u., $F_a^{\max} = 140$ MW, and $S_{\text{base}} = 100$ MVA. Thus the thermal limits and branch-flow relations already imply every default angle-difference bound. Removing these bounds preserves the hard feasible region while leaving the state and objective unchanged, even though the full-formulation task prescribes their construction.

This analytic example concerns the default branch angle-difference hard bounds under the frozen branch parameters and hard thermal limits. Instruction-specific changes to these parameters or limits require their own implication check. The reference-bus angle fixing is a separate requirement.

Incorrect constraints with an unchanged empty feasible region. For infeasible tasks, the automated evaluator retains the state and objective checks and requires solver-confirmed infeasibility. When these checks pass, different constraint implementations can still give $\mathcal{F}_C = \mathcal{F}_R = \varnothing$ while inducing infeasibility for different reasons. A constraint error that leaves the region empty therefore has zero effect on this comparison object. Agreement of state, objective, and infeasibility does not establish faithful implementation of the prescribed constraints. Review can inspect the implemented relations and determine whether the infeasibility follows from correctly realized task requirements.

I.4 ZERO-VALUED FIELDS AND PROPORTIONAL UPDATES

Proportional updates to zero-valued fields have zero numerical effect: $\alpha \cdot 0 = 0$. An incorrect multiplier, an omitted update, or an update applied outside the prescribed scope can therefore leave the returned values unchanged. When the state and induced formulation remain unchanged, the automated instance comparison cannot distinguish the incorrect operation.

For example, a proportional lower-output-bound update that incorrectly includes synchronous condensers leaves their $P_g^{\min} = 0$ unchanged. Likewise, all generator quadratic-cost coefficients in the 3 frozen networks are zero, so a proportional c_{2g} update leaves them at zero even when applied to excluded generators. In the observed scope errors, the multiplier is correct; the violation is the inclusion of synchronous condensers outside the non-slack active-power generator scope. These examples are instances of the same limitation of comparison on fixed numerical data.

In both full-formulation DC-OPF synthesis and the protocolized workflow, the LLM receives the instruction and modeling contract without access to the concrete network snapshot. Zero quadratic-cost coefficients therefore do not simplify the task presented to the LLM. Their effect is an expected limitation of automated comparison on a fixed instance: distinct implementations can induce the same evaluated objects. Fidelity review directly examines whether the implementation follows the prescribed entity scopes and update rules. Its fidelity judgments and the conclusions supported by those judgments remain valid despite this degeneracy. Section I.5 reports the review findings alongside the automated certificate outcomes. The evaluator's behavior on controlled nonzero-cost cases is reported in Section J.4.

I.5 CERTIFICATE OUTCOMES AND MODELING FIDELITY IN OUTPUT REVIEWS

The reviews below assess the full-formulation fidelity of complete tasks under Section I.1, including deterministic construction and standard-constraint completion for workflow outputs.

Full-output fidelity review of DeepSeek V4 Flash. The review covers all 236 outputs under the fidelity criterion above: 118 from full-formulation synthesis and the corresponding 118 workflow outputs. Table 29 cross-tabulates certificate outcomes with review judgments for both pathways.

Table 29: Certificate outcomes and modeling fidelity in the complete DeepSeek V4 Flash output review. Fidelity judgments concern the entire task; workflow judgments include deterministic construction.

	Full-formulation synthesis			Workflow		
Certificate outcome	Faithful	Unfaithful	Total	Faithful	Unfaithful	Total
Pass	96	5	101	118	0	118
Not passed	0	17	17	0	0	0
Total	96	22	118	118	0	118

All 17 outputs that failed the certificate checks are judged unfaithful. Among the accepted full-formulation synthesis outputs, 96 are judged faithful and 5 unfaithful. The 5 accepted candidates apply a non-slack-generator update to synchronous condensers, which the public contract excludes from that scope. They scale linear costs in `case118_A_009` and `case300_A_009`, the current lower-output bound in `case118_A_012`, the current upper-output bound in `case300_A_008`, and quadratic costs in `case300_A_010`. The affected coefficients or bounds on the excluded entities are zero. The extra state writes violate the requested scope but leave the returned state and induced mathematical objects unchanged. Their acceptance is consistent with the instance-equivalence criterion.

All 118 workflow outputs are judged faithful after deterministic construction. In 114 tasks, the generated JDIMP explicitly lists every standard constraint. In 4 tasks—`case30_A_003`, `case118_A_002`, `case118_A_003`, and `case300_A_002`—the public workflow's deterministic full-coverage fallback supplies default hard angle-difference constraints that the JDIMP does not repeat. This completion is expected workflow logic. In all 4 cases the completed hard constraints and

returned state match the strict reference, and no state-modification error was found. The complete task-level fidelity ledger is included in the accompanying release package.

Targeted review of zero-valued quadratic-cost update tasks. Table 30 reports a whole-task review of 100 outputs across 25 candidate packages for 4 quadratic-cost update tasks: 56 direct-code and 44 protocol outputs from the main comparison and supplementary experiments. These tasks are selected because their instructions proportionally update c_{2g} while all generator quadratic-cost coefficients in the frozen networks are zero. This selection targets conditions that can enrich certificate-passing but unfaithful outputs with zero-effect errors. The review uses the same full-formulation fidelity criterion as the DeepSeek V4 Flash review.

Table 30: Certificate outcomes and whole-task modeling fidelity for the 100 outputs selected for zero-valued proportional c_{2g} updates. Counts describe the targeted review set.

Certificate outcome	Faithful (review)	Unfaithful (review)	Total
Pass	66	11	77
Not passed	0	23	23
Total	66	34	100

Of the 11 accepted but unfaithful outputs, 9 incorrectly extend the c_{2g} update to excluded synchronous condensers, and 2 add those entities' fixed costs to the objective despite correctly implementing the c_{2g} update. The affected coefficients are zero, so these scope violations have zero numerical effect on the frozen instances. All 23 outputs that do not pass the certificate checks are judged unfaithful; all 44 protocol outputs are judged faithful after deterministic construction. The accompanying release package provides all 100 task-level review records; its source audit ledger retains the supporting code evidence.

Quantitative implications. Among outputs that pass the certificate checks, 5/219 (2.28%) are unfaithful in the complete DeepSeek V4 Flash review and 11/77 (14.29%) are unfaithful in the targeted review. All 5 DeepSeek V4 Flash discrepancies occur in full-formulation synthesis. Among outputs judged unfaithful, 5/22 (22.7%) and 11/34 (32.4%), respectively, still pass the certificate checks. The reviews thus quantify an observed distinction between certificate acceptance and modeling fidelity. These counts and percentages describe their respective review sets: the targeted set combines different pathways and related supplementary outputs, and overlaps the DeepSeek V4 Flash review.

The paper's conclusions draw jointly on the benchmark statistics and implementation reviews: the evaluated LLMs do not reliably perform end-to-end physical modeling in full-formulation DC-OPF synthesis, and assigning the interpretation of operational meaning to the LLM while assigning network-instance binding and complete DC-OPF construction to deterministic software improves modeling reliability. Benchmark pass rates and formal failure counts continue to use the automated certificate criterion. Review findings, including zero-effect violations, provide additional evidence for these conclusions and are reported separately from the automated failure counts.

J REFERENCE AND EVALUATOR VALIDATION

For modelable tasks, the evaluator expands the reference JDIMP into JDEF against a fresh network snapshot. A dedicated reference path then applies the state changes and assembles the affine constraints, soft semantics, and generation-cost coefficients used for evaluation. The evaluator reference path and the workflow share the network loader, public-identity mappings, topology and DC-parameter utilities, and the JDIMP-to-JDEF compiler. Appendix K breaks that coupling through reconstructions from raw MATPOWER data that do not import the production loader, compiler, model-construction helpers, or evaluator.

This appendix summarizes the validation evidence and documents the evaluator's assumptions, numerical procedures, reference-JDIMP review, and controlled evaluator tests. Detailed fidelity and formal-failure reviews appear in Sections I.5 and L.1, respectively.

J.1 VALIDATION OVERVIEW AND SUMMARY OF EVIDENCE

Reference review compares the JDIMPs with their natural-language instructions. Independent reconstruction, reported separately in Appendix K, checks reference interpretation and the production path from JDIMP through protocol expansion and workflow backend realization.

Output reviews assess certificate decisions against the declared automated criteria, including construction and interface requirements, instance-level formulation equivalence for feasible tasks within the specified tolerances, and the prescribed checks for other task outcomes (Section 4.1). The complete formal-failure review checks whether each rejected output violates these requirements and whether its rejection is justified. The complete DeepSeek V4 Flash output review additionally covers accepted outputs, checking for missed violations of the automated criteria as well as erroneous rejections. Table 31 summarizes this evidence.

Table 31: Validation of automated evaluation on the formal benchmark.

Validation evidence	Coverage	Validation check	Result
Reference JDIMP review	Formal 118-task benchmark	Instruction against structured reference intent	No recorded reference-interpretation error
Independent reconstruction	12 tasks selected for mechanism coverage	Raw MATPOWER data to independent Pyomo formulation; constraint, coefficient, coordinate, state, and objective comparison	Normalized coefficients agree to within 2.22×10^{-16}
Complete formal-failure review	244 full-formulation synthesis + 8 workflow failures	Whether output errors justify each rejection under the declared automated criteria	252/252 rejections supported by review; no erroneous rejection identified
DeepSeek V4 Flash full-output review	118 outputs from full-formulation synthesis and the corresponding 118 workflow outputs	Missed violations of automated criteria among accepted outputs; justification of failure decisions	No erroneous acceptance or rejection identified

The reference review identified no recorded reference-interpretation error. Across the 12 independent reconstructions, the production models agreed on normalized affine-constraint coefficients to within 2.22×10^{-16}. The complete formal-failure review found no erroneous rejection. The DeepSeek V4 Flash full-output review found no erroneous acceptance or rejection under the declared automated criteria. Appendix K reports the independent reconstructions. Section L.1 summarizes all formal failures. The complete one-line failure ledger and DeepSeek V4 Flash output review records are included in the accompanying release package. The whole-task fidelity findings are presented in Section I.5. Section J.4 supplements the benchmark evidence with controlled evaluator tests on nonzero quadratic costs.

J.2 EVALUATOR ASSUMPTIONS AND AUXILIARY VARIABLES

Assumptions used by the evaluator. As specified in Appendix B.3, the model variables are exactly the union of the returned `Pg`, `Theta`, and `Soft` maps. The hard feasible-region comparison uses $x = (P, \theta)$, aligned by public generator and bus identity after the network-state check. Hard constraints are affine in these coordinates, and branch flows are affine expressions in the bus angles. The permitted soft slacks s_k have prescribed epigraph forms and penalty coefficients, checked with the objective as described in Section 4.1. The interface excludes separate branch-flow variables

and arbitrary hidden auxiliaries. The following arguments concern hard affine feasibility in exact arithmetic; numerical tolerances are given in Appendix J.3.

Matching soft slacks to reference relations. The evaluator matches soft slacks by their live Pyomo structure rather than by variable names or enumeration order. For each slack, it collects the affine epigraph inequalities that contain that variable, converts a lower-bound Pyomo form to the same less-than residual convention when necessary, and expresses the remaining terms in the public P and θ coordinates; fixed public coordinates are substituted into the constant. A candidate slack can match a reference soft relation only if it is an unfixed continuous variable with lower bound zero and no finite upper bound, every associated epigraph contains exactly that slack with coefficient -1 and no unmapped variables, the public-variable support, coefficients, and constant agree with the reference residual, and its positive linear objective coefficient agrees with the prescribed penalty. Numeric comparisons use relative tolerance 10^{-9} and absolute tolerance $\max(10^{-10}, 10^{-9}\kappa)$, where κ is the comparison scale.

The evaluator forms all candidate–reference pairs satisfying these checks and requires a complete one-to-one assignment; extra, missing, or unmatched slacks therefore fail the interface check. In full-formulation synthesis, a one-sided soft relation uses one epigraph, while a soft equality uses the same slack in its two signed epigraphs. The protocolized workflow does not return a `Soft` map, so its auxiliary set is derived as all model variables not mapped through `Pg` or `Theta`; a soft equality is expanded into two directional reference rows to match the deterministic backend's two directional slacks. Only successfully assigned slack variables and their epigraph constraints are excluded from the hard-row comparison.

For the prescribed epigraphs and $\lambda_k > 0$, the shared-slack and directional-slack representations of a soft equality both induce the penalty $\lambda_k|h_k(x)|$ at each fixed x after minimization over their auxiliary variables. Coefficient-wise objective agreement is assessed between each candidate and its corresponding reference after soft-slack matching within the representation prescribed for that interface.

Conditions for direct bidirectional violation LPs. For the nonempty hard feasible regions of feasible tasks, a weaker sufficient condition is that both regions admit complete, finite affine descriptions in the same public semantic coordinates, possibly obtained through verified auxiliary elimination. To retain a polynomial guarantee relative to the supplied model size, both the cost of obtaining these descriptions and their size must be polynomial in that input size.

Under this condition, each test maximizes a target affine residual over a source polyhedron. For source X and target Y in $\{C, R\}$, the test in equation 3 is

$$V_{X\to Y}(j,d) = \max_{x\in\mathcal{F}_X} d\, r_j^Y(x), \qquad d \in \begin{cases}\{+1\}, & j \in \mathcal{I}_Y,\\ \{+1,-1\}, & j \in \mathcal{E}_Y.\end{cases}$$

Here $r_j^Y(x)$ is the target affine constraint residual, and $\mathcal{I}_Y$ and $\mathcal{E}_Y$ index the target inequalities and equalities, respectively. Requiring $V_{X\to Y}(j,d) \le 0$ for every target constraint and direction establishes $\mathcal{F}_X \subseteq \mathcal{F}_Y$. Applying both containment tests gives the criterion in equation 4:

$$\mathcal{F}_C = \mathcal{F}_R \quad\Longleftrightarrow\quad \mathcal{F}_C \subseteq \mathcal{F}_R \ \wedge\ \mathcal{F}_R \subseteq \mathcal{F}_C.$$

For example, consider auxiliary variables $y \in \mathbb{R}^q$ with a specified and verified affine definition $y = Tx + t$. With $H \in \mathbb{R}^{m\times n}$, $K \in \mathbb{R}^{m\times q}$, $h \in \mathbb{R}^m$, and $n = |G| + |B|$, direct substitution gives

$$\pi_x\{(x,y) : y = Tx + t,\ Hx + Ky \le h\} = \{x : (H + KT)x \le h - Kt\}, \tag{18}$$

where π_x denotes projection onto the public coordinates x. Sound elimination requires validating the defining equalities and applying the substitution consistently to every remaining constraint, variable bound, and objective term. An interface extension with variables declared as public branch flows would, for example, also need to certify each auxiliary variable y_a through the defining relation

$$y_a = P_a(\theta) = S_{\text{base}}\beta_a(\theta_{i_a} - \theta_{j_a}), \qquad a : i_a \to j_a, \tag{19}$$

with β_a derived from the transformed network state under the convention in equation 5. Each auxiliary variable y_a must map to a unique public branch identity and its prescribed source direction; endpoint buses alone do not identify a branch when parallel branches are present.

However, this weaker condition leaves the LLM responsible for constructing the correct physical relations and binding them to the intended entities. The evaluator would additionally need to identify and verify the auxiliary definitions and substitute them throughout the formulation, including checking the physical coefficients, units, and directions of variables declared as branch flows. We therefore adopt the stronger interface constraints above to make the model-construction obligations explicit and keep hard feasible-region certification within direct rowwise LP tests.

Arbitrary auxiliary variables and projected containment. Now retain the same public coordinates x but allow arbitrary continuous auxiliaries $y \in \mathbb{R}^q$, without a prescribed affine definition. After converting equalities and bounds to inequalities, write the candidate's lifted hard feasible region and its projection as

$$\begin{aligned} \widetilde{\mathcal{F}}_C &= \{(x, y) : Hx + Ky \le h\}, \\ \mathcal{P}_C &= \pi_x(\widetilde{\mathcal{F}}_C) = \{x : \exists y \in \mathbb{R}^q,\ Hx + Ky \le h\}, \end{aligned} \tag{20}$$

Equivalence in public coordinates would require $\mathcal{P}_C = \mathcal{F}_R$, with both containments tested. For each reference inequality $a_j^\top x \le b_j$, forward containment can still be checked by the LP

$$\sup_{(x,y)\in\widetilde{\mathcal{F}}_C} (a_j^\top x - b_j) \le 0. \tag{21}$$

Reverse containment, however, requires a feasible auxiliary assignment for every reference-feasible public point:

$$\mathcal{F}_R \subseteq \mathcal{P}_C \quad \Longleftrightarrow \quad (\forall x \in \mathcal{F}_R)(\exists y \in \mathbb{R}^q)\ Hx + Ky \le h. \tag{22}$$

The auxiliary assignment can depend on x and must jointly satisfy all candidate inequalities. The lifted candidate rows therefore do not directly provide the target inequalities in public coordinates needed by the reverse tests in Section 4.2.

For rational polytopes in variable dimension, this reverse-containment problem is co-NP-complete (Kellner, 2015, Theorem 2.4). Consequently, a polynomial-time exact test for arbitrary lifted polyhedral formulations would imply $\mathrm{P} = \mathrm{NP}$. The current interface keeps both hard feasible-region containment directions within the direct rowwise LP procedure in Section 4.2.

Public semantic identities and admissible coordinates. Even if the projected containment problem is solved exactly, the comparison still requires a specified map from candidate coordinates to public physical quantities. In particular, the projection π_x above presupposes that the public quantities $x = (P, \theta)$ have been identified. Public generator and bus IDs identify physical entities; backend row keys, variable names, and vector positions are access coordinates. Even two otherwise interchangeable generators with identical parameters retain distinct identities when an instruction names a specific device. An internal coordinate permutation is admissible together with a map that preserves these identities.

For example, let $P \in \mathbb{R}^{|G|}$ collect generator active-power outputs in the public coordinate ordering, and let $\widehat{P} = \Pi P$ contain the same outputs in the candidate's internal ordering. The permutation matrix $\Pi \in \{0, 1\}^{|G|\times|G|}$ has exactly one 1 in each row and column and satisfies $\Pi^{-1} = \Pi^\top$. A bound requested for public generator $g_\star$ becomes

$$e_{g_\star}^\top P \le P_{g_\star}^{\max} \quad \Longleftrightarrow \quad e_{g_\star}^\top \Pi^\top \widehat{P} \le P_{g_\star}^{\max}. \tag{23}$$

where $e_{g_\star}$ selects that generator in the public coordinate ordering. The same identity requirement applies to instruction-induced state changes, including when parameter symmetry makes an identity substitution numerically invisible. Thus, permitting coordinate permutations or auxiliary variables still requires public semantic alignment before mathematical comparison.

J.3 Constraint normalization and numerical tolerances

Reference and candidate constraint construction. The reference and protocolized candidate paths share the deterministic JDIMP-to-JDEF compiler. After JDEF, the reference path directly assembles affine constraints, soft-constraint residuals, and generation-cost coefficients for the evaluator. The workflow constructs a live Pyomo model, from which the evaluator extracts candidate constraints. These branches use separate routines to apply state changes and assemble the formulation, while reusing the network loader, topology utilities, and DC power flow components. The normalization below applies to the resulting reference and candidate constraints.

Homogeneous constraint normalization. After expressing hard constraints, variable bounds, and fixed values in the public coordinates, the evaluator writes each unnormalized affine constraint residual for $X \in \{C, R\}$ as $\widetilde{r}_j^X(x) = (a_j^X)^\top P + (b_j^X)^\top \theta + c_j^X$, with constraint $\widetilde{r}_j^X(x) \le 0$ or $\widetilde{r}_j^X(x) = 0$. Each side of a ranged constraint becomes a separate inequality. The normalized residual used in Section 4.2 is

$$\gamma_j^X = \max\left(|c_j^X|, \|a_j^X\|_\infty, \|b_j^X\|_\infty\right),$$
$$r_j^X(x) = \begin{cases} \widetilde{r}_j^X(x)/\gamma_j^X, & \gamma_j^X > 0, \\ 0, & \gamma_j^X = 0. \end{cases} \tag{24}$$

The divisor has no unit lower bound. Only exact zero coefficients are removed before division. An identically zero constraint residual defines a tautology; a nonzero constant constraint residual normalizes to $+1$ or -1, so a positive constant inequality or a nonzero constant equality is contradictory. Nonfinite entries and underflow of a nonzero term during normalization produce a numerical error. The original coefficients and divisor are retained with each constraint. The constraint residuals recorded in certificates use r_j^X; the constraint residual in the original units is $\widetilde{r}_j^X(x) = \gamma_j^X r_j^X(x)$.

Constraint normalization applies to reference and candidate hard constraints, candidate optimization on a private model copy, both containment LPs, certificates, and witness feasibility checks. Variable units and objective coefficients are preserved. In the solve copy, each validated soft epigraph is scaled as a complete affine constraint, including its slack coefficient. External soft-penalty evaluation uses the soft-constraint residual in physical units and its prescribed penalty coefficient.

Containment thresholds and witness confirmation. Every normalized hard constraint has unit tolerance scale, giving

$$\tau = \max(\epsilon_{\text{abs}}, \epsilon_{\text{rel}} \cdot 1) = 10^{-7}, \qquad \epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 10^{-7}. \tag{25}$$

For each direction d in equation 3, the numerical test uses $V_{X\to Y}(j, d) = \max_{x\in\mathcal{F}_X} d\, r_j^Y(x)$. An optimal value at most τ passes that directed test. A value in $(\tau, 10\tau]$ lies in the numerical gray zone. A value above $10\tau = 10^{-6}$ yields a failure certificate only after independent confirmation of a finite witness. The directed normalized constraint residual for the target constraint must exceed 10τ, and every constraint and bound defining the source feasible region must have normalized violation at most 10τ. Equality violations use absolute constraint residuals; inequality violations use the positive parts of constraint residuals. For an unbounded violation LP, an auxiliary feasibility solve seeks a finite witness satisfying $d\, r_j^Y(x) \ge 20\tau$, followed by the same confirmation. An unresolved result in either containment direction, including an outcome in the numerical gray zone or failed witness confirmation, is recorded as `EVALUATOR_ERROR`.

Solver settings. HiGHS uses primal and dual feasibility tolerances of 10^{-9}, a small-matrix coefficient threshold of 10^{-12}, and a 120-second limit per solve. An LP returning `unknown` or `error` is retried once at primal and dual tolerances of 10^{-8}, using the interior-point method for candidate optimization and simplex for containment. Convex quadratic candidate objectives use the quadratic solver interface with the primary tolerances. Unresolved termination, timeout, solve exceptions, and nonfinite solve or constraint residual values are recorded as numerical errors.

State and objective comparisons. Objective coefficients are expressed in the aligned public and soft coordinates described in Appendix J.2. Numeric scalar state fields and canonical objective coefficients are compared using $|u - v| \le 10^{-9}\max(1, |u|, |v|)$. For solved-objective comparisons, let J_R be the reference optimum, J_E the externally reconstructed cost at the candidate dispatch, and J_C the candidate-declared objective. The checks require

$$|J_E - J_R| \le \max\left(10^{-3}, 2\times 10^{-7}\max(1, |J_R|)\right), \tag{26}$$
$$|J_C - J_E| \le \max\left(10^{-3}, 2\times 10^{-7}\max(1, |J_C|, |J_E|)\right). \tag{27}$$

These comparisons use the declared state and cost units. The separate reference-reconstruction checks report their own comparison tolerances.

J.4 Controlled validation with nonzero quadratic costs

The frozen case30, case118, and case300 data contain 6, 54, and 69 polynomial cost rows, respectively, all with $c_{2g} = 0$. The nonzero-cost tests examine whether the evaluator accepts correct formulations

and detects objective errors at other coefficient values. We use controlled formulations with known correct objectives and deliberate perturbations. Table 32 summarizes the coefficient checks, an analytic quadratic program, and complete network evaluations. The tests use Pyomo 6.9.5, and HiGHS through highspy 1.12.0. Convex quadratic objectives are solved through Pyomo's `highs` interface; containment checks remain LPs over affine hard constraints.

Objective-coefficient tests. The synthetic model has $P_1 + P_2 = 10$, $0 \le P_1, P_2 \le 10$, and objective

$$f(P) = q_1 P_1^2 + q_2 P_2^2 + 3P_1 + 4P_2 + 12. \tag{28}$$

A third generator output and a reference angle are fixed at zero to test terms involving fixed public coordinates. The 4 coefficient pairs are $(q_1, q_2) = (0.01, 0)$, $(0.01, 0.03)$, $(2.5, 0.75)$, and $(10^{-8}, 3 \times 10^{-8})$. For each pair, both the original objective and an algebraically equivalent rewrite pass coefficient comparison. Each pair is also tested with 10 perturbations: omission of all or one quadratic term; incorrect magnitude, sign, generator assignment, or unit scale; an added cross-generator term; a generator–angle cross term involving the fixed reference angle; an extra quadratic term on the fixed generator; and addition of $0.1(P_1 - P_1^\star)^2$, where $P_1^\star$ is the analytic reference optimum. The last perturbation preserves the optimal dispatch and value while changing the objective polynomial. All 40 perturbations fail coefficient comparison. With the existing coefficient tolerances, an increment of 5×10^{-10} to $q_1 = 0.01$ is accepted, whereas 2×10^{-9} is rejected.

Analytic solve and network tests. For $(q_1, q_2) = (2.5, 0.75)$ in equation 28, the analytic solution is $(P_1^\star, P_2^\star) = (16/6.5,\ 10 - 16/6.5)$ with objective 107.3076923076923. The quadratic solver matches the dispatch within 10^{-6} and the objective within 10^{-8}.

The complete evaluator is tested on nonzero-cost variants of `case30_A_001`, `case118_A_001`, `case300_A_001`, and `case30_A_040`; the last includes a bilateral soft branch-flow limit. Each variant assigns $c_{2g} = 0.01$ to all active-power decision generators and reconstructs its reference coefficients and optimum. All 4 correct controls pass the complete evaluation, including quadratic optimization, external objective checks, soft-constraint checks where applicable, and bidirectional containment. For each variant, we then omit one quadratic term, increase one quadratic coefficient by 50%, reverse its sign, or add a cross-generator term. All 16 erroneous formulations are rejected before candidate optimization: 12 by coefficient comparison and 4 sign reversals by the convexity check.

Table 32: Controlled evaluator validation with nonzero quadratic costs. Counts refer to constructed test cases.

Test group	Observed result
Correct and equivalent synthetic objectives	8/8 accepted
Synthetic objective perturbations	40/40 rejected by coefficient comparison
Coefficient-tolerance boundary	2/2 decisions as expected
Analytic convex quadratic program	Dispatch and objective match analytic values
Correct nonzero-cost network formulations	4/4 pass the complete evaluator
Perturbed network objectives	16/16 rejected before optimization: 12 by coefficients, 4 by convexity

Interpretation. The tests identify no detection blind spot for the constructed nonzero-cost cases: correct controls pass and the specified objective errors are detected. This evidence concerns the tested coefficient values and perturbations. The network tests reuse the evaluation-side reference construction path and assess integration of coefficient comparison, quadratic optimization, and containment. Diagnostic network variants and their recomputed reference optima remain separate from the 118-task benchmark and its reported scores.

Because the LLM receives neither the network snapshot nor its cost coefficients, the zero values of c_{2g} do not simplify the information available to the LLM or relax its implementation obligations. The generated program must still construct the objective and apply requested coefficient updates to the values supplied at execution. The degeneracy instead affects the observability of implementation errors on the frozen instances: an incorrect proportional update can have zero numerical effect when the original coefficient is zero. A zero-effect error that preserves the compared state, objective, soft

epigraphs, and hard feasible region satisfies the instance-equivalence criterion. Acceptance in this situation is the expected outcome of that criterion. The whole-task fidelity reviews in Section I.5 inspect implementation requirements directly, including errors whose numerical effects are masked by zero-valued fields (Section I.4).

K Independent Validation of the JDIMP-to-Pyomo Construction Path

Coverage and selection. This audit validates the production construction path from a reviewed reference JDIMP through deterministic JDIMP-to-JDEF compilation and workflow backend processing to the live Pyomo model. The independent reconstruction also cross-checks the reference interpretation against the instruction. At the level of construction mechanisms, the sample substantially covers the modelable task types tested by the benchmark (Table 8). The audit comprises 12 tasks from the 118-task benchmark: 7 from case30, 3 from case118, and 2 from case300, with 11 optimal outcomes and 1 infeasible outcome. Tasks were selected before reconstruction to cover distinct construction mechanisms.

The 3 base instances exercise all four standard constraint families—nodal balance, generator limits, thermal limits, and angle-difference limits—as well as variable bounds, reference-angle fixing, and generation costs across the 3 network sizes and their line, transformer, and impedance representations. The 9 task-specific instances cover network and parameter changes through branch disconnection, load and raw-reactance updates, and current-output updates; generator dispatch, coordination, and reserves through updated-state deviation limits, pairwise and group relations, and zone-based remaining capacity; and interface and security requirements through source-oriented flow aggregation, bilateral soft flow bands, and selective soft thermal limits. These instances also include analytically certifiable infeasibility. Together these instances check entity selection, state application, constraint direction, attribute aggregation, and soft penalties, including combinations of these operations. For each task, the audit compares the entire resulting formulation, including constraints and soft slacks that are inactive at the optimum.

Reconstruction and comparison. A separate implementation reconstructs each formulation from a fixed manual interpretation of the corresponding instruction and raw MATPOWER data. The independent implementation does not consume the reference JDIMP or JDEF, or import the project loader, compiler, model-construction helpers, or evaluator. A comparison harness runs the reference JDIMP through the compiler and workflow backend to obtain the production Pyomo model used in this audit. It separately checks instruction-to-JDIMP interpretation and JDEF expansion, then compares the independent and production models in stable public coordinates, including semantic coordinates for soft slacks. Checks cover the modeled bus, generator, and branch state, every variable coordinate and objective coefficient (including soft penalties), and every normalized affine constraint, variable bound, and reference fix. Coefficient and state comparisons use tolerance 2×10^{-9}, and solved-objective comparisons use 2×10^{-6}. Both paths use Pyomo and HiGHS; their solved outcomes supplement the structural comparisons. The infeasible instance is compared structurally before its termination status is checked and is also verified by an analytic infeasibility certificate.

Table 33: Independent reconstruction of 12 reference formulations. Variable counts list generation, angle, and, where applicable, soft-slack coordinates. Affine constraints include variable bounds and reference fixes.

Task	Coverage	Variables	Affine constraints	Independent / production / frozen objective	Max. $\lvert\Delta P_g\rvert$
`case30_A_001`	case30 base	$2 + 30$	199	7472.814670372456 / 7472.814670372476 / 7472.814670372476	3.41×10^{-13}
`case118_A_001`	case118 base	$19 + 118$	901	93100.72992629727 / 93100.72992629746 / 93100.72992629746	3.64×10^{-12}
`case300_A_001`	case300 base	$57 + 300$	2059	517851.07520212274 / 517851.07520212256 / 517851.07520212256	4.82×10^{-11}
`case30_A_022`	Bus-state update	$2 + 30$	199	8380.363788963934 / 8380.363788963941 / 8380.363788963941	1.14×10^{-13}
`case30_A_029`	Directed interface constraint	$2 + 30$	200	8036.305583599972 / 8036.3055835999985 / 8036.3055835999985	4.97×10^{-13}
`case30_A_040`	Bilateral soft flow	$2 + 30 + 2_{\mathrm{soft}}$	203	513359.63862993475 / 513359.63862993574 / 513359.63862993574	3.41×10^{-13}
`case30_A_036`	Impedance x update	$2 + 30$	199	7472.536157188217 / 7472.536157188208 / 7472.536157188208	1.71×10^{-13}
`case30_A_042`	Updated-state reference	$2 + 30 + 1_{\mathrm{soft}}$	201	66019.20155382791 / 66019.20155382950 / 66019.20155382950	3.13×10^{-13}
`case30_A_017`	Analytic infeasibility	$2 + 30$	200	infeasible / infeasible / infeasible	–
`case118_A_027`	Outage and selective soft limits	$19 + 118 + 4_{\mathrm{soft}}$	901	93100.73959403516 / 93100.7395940352 / 93100.7395940352	1.19×10^{-12}
`case300_A_052`	Zone remaining capacity	$57 + 300$	2060	523623.6425124493 / 523623.6425124559 / 523623.6425124559	1.63×10^{-10}
`case118_A_054`	Bus-to-generator binding	$19 + 118$	902	94572.80912822096 / 94572.80912822112 / 94572.80912822112	1.11×10^{-11}

Reconstruction results. The production and independently reconstructed formulations agree to a maximum absolute difference of 2.22×10^{-16} in normalized affine-constraint coefficients. The independent and production hashes of normalized constraints and objective polynomials match instance by instance; constraint hashes use coefficients rounded to 9 decimal places, with unrounded differences checked separately. The largest solved-objective absolute difference is 6.58×10^{-9}. For the 3 base cases, the balance/generator/thermal/angle specification counts are $30/2/41/41$, $118/19/186/186$, and $300/57/411/411$, respectively. The state-change instance raises bus 7 and bus 8 loads from 22.8 and 30.0 MW to 32.8 and 40.0 MW. The interface instance adds the source-oriented constraint $P_1(\theta) + P_2(\theta) \leq 200$ MW, increasing the number of hard constraints in case30 from 199 to 200. The bilateral soft-flow instance adds the band $-16 \leq P_1(\theta) \leq 16$ MW with a penalty coefficient of 5000 per MW on each slack; its independent optimum uses 101.006598 MW of upper slack and 0 lower slack. The impedance instance independently identifies source branches 13, 14, and 16 and increases their raw series reactances by 5%, to 0.2184, 0.1155, and 0.1470 p.u. In the updated-state instance, generator 2's current output is first changed from 46 to 49 MW, so the subsequently evaluated soft upper bound is 55 MW; the independent optimum uses 11.709277 MW of slack. For the infeasible instance, $P_1 \leq P_2 + 20$ MW and $P_2 \leq 92$ MW imply total generation of at most 204 MW, whereas summed power balance requires 283.4 MW, yielding a 79.4 MW infeasibility certificate.

In `case118_A_027`, disconnecting source branch 46 leaves 185 active branches. Only branches 9 and 32 receive soft thermal limits, leaving 183 hard-limited branches. The soft limits on branches 9 and 32 are 710 and 768 MW, respectively, and each direction has a separate nonnegative slack with penalty coefficient 5000 per MW. All 4 optimal slacks are 0, but their constraints, bounds, and objective coefficients are checked explicitly. This complements the bilateral soft-flow instance with a nonzero optimal slack.

For `case300_A_052`, selecting zone 2 from the raw bus records and then binding its online active-power generators yields 80 buses and 18 generators, with total maximum generation of 14442 MW. The requirement of at least 6000 MW remaining capacity therefore becomes an aggregate generation upper bound of 8442 MW, attained at the optimum. For `case118_A_054`, raw generator records map buses 10 and 12 to source generators 5 and 6. The group requirement is $P_5 + P_6 \geq 520$ MW; the optimal outputs are 505 and 15 MW. Both instances match the reference entity sets, attribute sums, inequality directions, and complete model coefficients. Across the selected reconstructions, the state, coordinate, objective, and complete normalized-constraint comparisons support the production JDIMP-to-Pyomo construction path for the exercised mechanisms.

L Failure Analysis

This appendix presents the complete review of formal failures, a containment failure with matching optimal values, and the paired analysis of workflow improvements. The complete 252-record main-comparison failure ledger is included in the accompanying release package.

L.1 Complete review of formal failures: summary

Every formal rejection in the main 118-task comparison was reviewed: 244 outputs from full-formulation synthesis and 8 workflow outputs, for 252 model–task failures in total. Diagnoses of full-formulation synthesis failures use first-error attribution. A workflow failure is attributed to the end-to-end generated formulation or certificate decision. All 252 records contain a candidate or output error; none is an erroneous rejection caused by the reference or evaluator.

Table 34: Coverage of the complete formal-failure review.

Model	Full-formulation synthesis	Workflow	Total
Claude Opus 5	14	1	15
Claude Sonnet 5	29	1	30
DeepSeek V4 Flash	17	0	17
DeepSeek V4 Pro	21	0	21
Gemini 3.1 Pro Preview	23	0	23
Gemini 3.6 Flash	40	1	41
GPT 5.6 Luna	65	1	66
GPT 5.6 Sol	22	3	25
Kimi K3	13	1	14
Total	244	8	252

Table 35: First failed evaluator check across the 252 formal failures.

First failed evaluator check	Count
Semantic-translation and structured-output check	3
Build and static-validity check	44
Outcome routing (feasibility and typed rejection)	83
Public-state equality check	59
Public return-interface check	19
Bidirectional violation-LP check	26
Soft-semantics check	18
Total	252

Outcome routing combines 52 candidates that report infeasibility for a nonmatching construction, 20 typed rejections of modelable requests, and 11 missing typed rejections. The violation-LP table row reports the bidirectional containment stage as one check and is not partitioned by certificate direction.

Table 36 summarizes selected error subcategories, their assignments to the four main-text categories, and their counts under the same first-error attribution as the main text.

Table 36: Selected error subcategories among the 244 formal failures in full-formulation synthesis.

Error subcategory	Main-text category	Count
Missing system-base MW scaling	Physical relations	38
Writing dispatch conditions into public state	Constraint semantics	35
Incorrect impedance element-base conversion	Physical relations	21
Synchronous-condenser scope pollution	Entity binding and scope	20
Branch-flow direction or incidence-sign errors	Physical relations	18
Noncanonical return-key encoding	Program implementation	17
Invalid imports	Program implementation	13
Case metadata or name mismatches	Program implementation	11
Generator-domain errors in public-variable maps	Entity binding and scope	2

A paired analysis of workflow improvements appears in Section L.3; the complete ledger of all 252 formal failures, including 8 workflow failures, is included in the accompanying release package.

L.2 Containment failure with matching optimal values

Task `case30_A_029` requires the combined flow on branches 1 and 2, measured in their public directions, to be at most 200 MW. GPT 5.6 Luna omits the factor S_{base}/S_a when converting equivalent-impedance parameters from the element MVA base S_a to the system base. The omitted conversion changes branch-flow coefficients and nodal-balance equations while preserving the generation-cost objective. The candidate-declared objective, externally reconstructed objective, and reference optimum nevertheless agree at $8{,}036.3056$ within tolerance. The forward LP finds a candidate-feasible witness violating the reference balance at bus 10 by 35.4584 MW; the reverse LP finds a reference-feasible witness violating the candidate balance at the same bus by 17.6235 MW. The forward and reverse LP witnesses establish that the hard feasible regions fail containment in both directions despite the matching optimal cost.

L.3 Paired analysis of workflow improvements

We pair the two certificate outcomes for each model and task. There are 242 pairs in which full-formulation synthesis fails and the workflow passes. Table 37 groups these improvements by the confirmed first error in the full-formulation output, using four mutually exclusive categories organized by modeling responsibility. The shares describe the composition of improvements under the complete workflow. The table also retains all 244 full-formulation failures as denominators for the corresponding workflow pass rates.

Table 37: Reviewed causes of paired workflow improvements. S denotes full-formulation synthesis and W the workflow. Paired W passes count S-fail/W-pass pairs. Share uses all 242 such pairs as its denominator; W pass rate uses the S failures in the same row.

First-error category	S failures	Paired W passes	Share (%)	W pass rate (%)
Physical relations	90	90	37.19	100.00
Entity binding and scope	49	49	20.25	100.00
Constraint semantics	36	34	14.05	94.44
Program implementation	69	69	28.51	100.00
Total	244	242	100.00	99.18

Category definitions. The physical-relations category covers the parameters, coefficients, and standard network equations and constraints used in the formulation. It comprises system-base MW/per-unit conversion (38), impedance element-base conversion (21), flow direction and incidence signs (18), physical parameter reconstruction after transformer-reactance updates (7), omitted standard angle constraints or reference-bus balance (4), a series-admittance formula error (1), and fabrication of missing new-line electrical parameters (1).

Entity binding and scope covers entity references, decision-domain membership, and the applicability and scope of requested operations and constraints. It comprises public-identity/backend-row confusion (12), generator-domain or update/constraint-scope errors (25), accepting non-unique branch references (3), accepting operations on devices outside the active-power decision domain (8), and rejecting a valid request because an unrelated synchronous condenser exists (1).

The constraint-semantics category covers the distinction between state changes and dispatch constraints, and between hard requirements and soft preferences. It comprises writing dispatch conditions into public state (35) and encoding a soft condition as hard without a state write as the first error (1). Compound diagnoses that both modify state and harden a soft condition are attributed to the state write when it is the first error. Of these 36 failures, 34 have passing workflow counterparts: 33 state/constraint errors and 1 hard/soft error.

Program implementation comprises Python/Pyomo/API, type, reference, and initialization errors (20), record-field construction or access errors (8), invalid imports (13), noncanonical return-key encoding (17), and case metadata or name mismatches (11).

All 90 physical-relations, 49 entity/scope, and 69 program-implementation failure pairs have passing workflow counterparts.

Distribution across models. Table 38 gives the same classification for each underlying LLM. Each of the four categories occurs among improved pairs for all 9 models.

Table 38: First-error categories for the 242 S-fail/W-pass pairs, by model. Relations denotes physical relations; Entity denotes entity binding and scope; Semantics denotes constraint semantics; Program denotes program implementation.

Model	Relations	Entity	Semantics	Program	Total
Claude Opus 5	6	3	1	4	14
Claude Sonnet 5	16	3	7	2	28
DeepSeek V4 Flash	4	5	5	3	17
DeepSeek V4 Pro	9	5	5	2	21
Gemini 3.1 Pro Preview	1	8	5	9	23
Gemini 3.6 Flash	2	7	5	26	40
GPT 5.6 Luna	36	12	1	16	65
GPT 5.6 Sol	12	4	2	3	21
Kimi K3	4	2	3	4	13
Total	90	49	34	69	242

Relation to the aggregate comparison. Table 39 retains all four paired outcomes. Of the 244 full-formulation failures, 242 have passing workflow counterparts (99.18%). Certificate checks fail in both settings for 2 pairs, and 6 pairs pass under full-formulation synthesis but fail under the workflow. Thus, the net reduction in certificate failures is $242 - 6 = 236$, yielding the 244-to-8 comparison and the 96.72% relative failure-rate reduction.

Table 39: Paired certificate outcomes for all 1,062 model–task combinations. Each model is evaluated on the same 118 tasks; S denotes full-formulation synthesis and W the workflow.

Model	S pass W pass	S fail W pass	S pass W fail	S fail W fail
Claude Opus 5	103	14	1	0
Claude Sonnet 5	89	28	0	1
DeepSeek V4 Flash	101	17	0	0
DeepSeek V4 Pro	97	21	0	0
Gemini 3.1 Pro Preview	95	23	0	0
Gemini 3.6 Flash	77	40	1	0
GPT 5.6 Luna	52	65	1	0
GPT 5.6 Sol	94	21	2	1
Kimi K3	104	13	1	0
Total	812	242	6	2

Both pairs that fail in both settings fall under constraint semantics, specifically state/constraint confusion in full-formulation synthesis. In `case118_A_039`, Claude Sonnet 5 writes a dispatch cap into public state under full-formulation synthesis and encodes the soft cap as hard under the workflow. In `case118_A_052`, GPT 5.6 Sol writes a dispatch lower bound into public state under full-formulation synthesis and emits no JDIMP JSON under the workflow. The 6 new workflow failures comprise 4 JSON-output failures and 2 hard/soft semantic errors. Across all 8 workflow failures, 5 concern JSON completeness or syntax and 3 concern hard/soft semantics.