arXiv:2601.00880v1 [cs.AI] 31 Dec 2025

通用条件逻辑：一种用于提示工程的形式语言

Fo基础规范、核心验证与研究路线图 p

Anthony Mikinka 独立研究者amikinka@asu.eduORCID：0009-0005-2955-4140

2025年12月30日

摘要

我们提出通用条件逻辑（UCL），这是一种用于提示优化的数学框架，可将提示工程从启发式实践转变为系统优化。通过系统评估（N=305、11 个模型、4 次迭代），我们证明其可显著减少令牌（29.8%，$t(10) = 6.36, p < 0.001$，Cohen's $d = 2.01$），并相应节省成本。UCL 的结构开销函数 $O_s(\mathcal{A})$通过过度指定悖论解释特定版本的性能差异：超过阈值 $S^* = 0.509$后，额外的规格会导致性能呈二次下降。核心机制——指示函数（$I_i \in \{0, 1\}$）、结构开销（$O_s = \gamma \sum \ln C_k$）和早期绑定——均得到验证。值得注意的是，最佳 UCL 配置因模型架构而异：某些模型（如 Llama 4 Scout）需要进行特定版本适配（V4.1）。本研究确立了 UCL 作为一种可校准框架，可用于实现高效的大语言模型交互；特定模型系列优化是一个重要的研究方向。

关键词： 提示工程、大语言模型、形式语言、编程范式、优化理论、领域特定语言

目录

1 引言 4

1.1 提示编程范式 .4 1.2 过度指定悖论 .4 1.3 结构开销验证 .5 1.4 指示函数机制 .6 1.5 贡献 .7 **2 Rela**

述工作 7

2.1 提示工程方法 .7 2.2 编译器与编程的相似之处 .8 **3 数**

学框架 8

3.1 通用提示方程 .8 3.2 质量函数 .9 3.2.1 惩罚机制推导 .10 3.3 结构开销 .11 3.4 拉格朗日优化 .12 3.4.1 卡鲁什-库恩-塔克条件 .12 **4 UCL**

核心语言规范 1

4.1 经验证的构造 .13 4.1.1 CONDITION 块（指示函数） .13 4.1.2 概念引用 .13 4.1.3 CRITICAL 指令（早期绑定） .13 4.2 形式文法（经验证的子集） .14 4.3 SWITCH 为何失效 .14

5 实证验证 15

5.1 实验设计 .15 5.2 结果 .16 5.3 统计分析 .19 5.3.1 自由度解释 .19 5.4 理论预测得到验证 .20 **6 讨论**

讨论 2

6.1 核心发现 .21 6.2 统计解释 .21 6.2.1 为什么 $t(10)$ 个模型？ .21 6.2.2 效应量解释 .21 6.2.3 与过度指定悖论的关系 .21 6.3 模型架构考量 .22 6.4 扩展规格预览 .23 6.5 验证路线图 .23 6.5.1 架构感知扩展 .23 6.6 演化中的语言 .24 6.7 编程范式的影响 .24 6.8 局限性与可推广性 .24

7 结论 **25**

A 附录 A：变量参考 **28**

A.1 核心符号 .28A.2 UCL 算子 .28A.3 提示词版本比较 .29A.4 语法示例 .29A.5 函数 .30A.6 经验确定的常数 .30A.7 完整 O_s 计算示例 .30A.7.1 V3 架构分析 .30A.7.2 计算 .30

A.7.3 V4比较 .31

B 附录 B：模式库 **31**

B.1 模式 1：条件域激活 .31B.2 模式 2：关键输出指令 .32B.3 模式 3：概念引用 .32B.4 反模式（已验证的失效） .33B.5 设计指南（经实证验证） .33B.6 局限性 .33

C语言扩展运算符与未来语法 **34**

D 研究 架构路线图：迈向静态分析 3 **4**

E 验证协议 **35**

1 引言

1.1 提示词编程范式

计算机发展史展现了抽象层次不断提升的历程：从机器码到汇编语言，从汇编语言到C语言，从命令式范式到声明式范式。大语言模型代表着下一个前沿——执行自然语言指令 作为代码的系统。然而，提示工程在很大程度上仍依赖启发式方法，缺乏形式文法或系统优化。

本文介绍了通用条件逻辑（UCL）这一形式语言，它将自然语言转换为针对大语言模型优化的可执行结构。正如 C语言将人类语法（`if`、 `while`）编译为高效指令一样，UCL 提供了一种具有明确以下要素的领域特定语言：

- **语法**：生成格式正确提示的产生式规则
- **句法**：算子（`^^CONDITION:^^`、 `[[LLM:]]`、{{concept:domain:spec}}）
- **语义**：将句法映射至行为的指示函数
- **语用学**：高效构建的设计原则

这使得系统优化成为可能，将提示工程从技艺转变为科学。

1.2 过度指定悖论

传统观点认为，增加规格说明会带来单调收益。我们的研究揭示了一种反直觉的现象：

Prompt quality is non-monotonic in specification level.

超过 $S^* \approx 0.509$ 后，更多细节会通过三种惩罚项降低质量：

$$Q(S) = \begin{cases} \frac{Q_{\max}}{S^*} S & \text{if } S \leq S^* \\ Q_{\max} - b(S - S^*)^2 & \text{if } S > S^* \end{cases} \tag{1}$$

其中 $Q_{\max} = 1.0$, $b = 4.0$。这与软件开发中的过度工程化如出一辙：过多的注释会增加维护负担。在提示词中，过度指定会引发认知泄漏——模型输出导航逻辑，而非解决方案。

三种惩罚机制：

1. 角色混淆（$P_{\text{role}} = \alpha_1 S^2$）：二次。证据：V2 在 $S = 0.40$achieved$Q = 0.02$（98% 失败率）。2. 认知复杂度（$P_{\text{complexity}} = \alpha_2 O_s$）：线性。证据：V3 配合 $O_s = 28.85$showed $4\times$ 出现了令牌膨胀。3. 感知复杂度（$P_{\text{perceived}} = \alpha_3 \log |P|$）：对数。证据：V4（142 行）出现了格式失败。

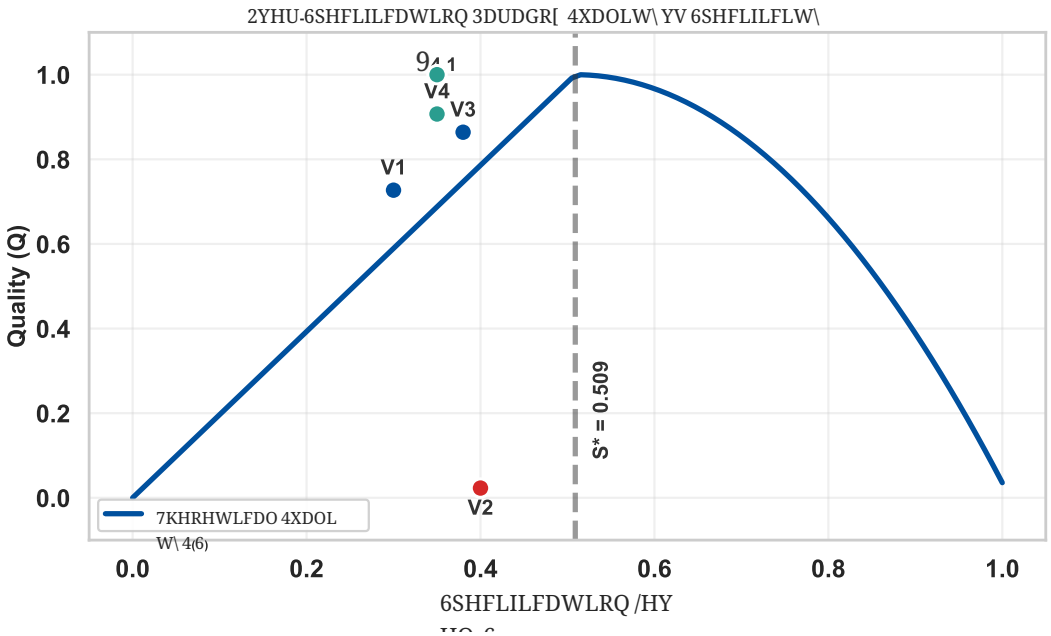


图1：过度指定悖论：非单调质量函数 $Q(S)$。质量随规格（$Q = 1.96S$）线性提升，直至达到最佳阈值 $S^* = 0.509$；超过此阈值后，额外的规格会导致二次退化（$Q = 1.0 - 4(S - S^*)^2$）。UCL 版本以标记表示：V4 和 V4.1（蓝色、紫色）在接近最优点时达到高质量，而 V2（红色）尽管规格增加，仍出现灾难性失效，展现了这一悖论。

1.3 结构开销验证

结构开销函数 $O_s(\mathcal{A})$ 针对每个版本进行计算，使用 g:

$$O_s(\mathcal{A}) = \gamma \sum_{k \in \mathcal{K}} \ln(C_k) + \delta |L_{\text{proc}}| \tag{2}$$

与 $\gamma = 1.0$ 和 $\delta = 0.1$。

表1：结构开销组成部分与质量

版本	架构	K	$\gamma \sum \ln(C_k)$	$\delta \lvert L \rvert$	O_s	质量
V1	2 SWITCH (8+4)	2	3.47	5.0	8.47	72.7%
V2	嵌套 SWITCH + COND	2	3.47	15.0	35.47	84.1%
V3	2 SWITCH + 无条件	2	3.47	10.0	13.47	86.4%
V4	7 个关键词条件	0	0.00	2.0	2.00	90.7%
V4.1	关键词 + [[CRITICA	0	0.00	2.0	2.00	100.0%

关键观测值：

- V1–V3 使用 SWITCH 架构：无论输入是什么，都会解析所有分支情形
- V4–V4.1 使用关键词条件：仅激活匹配的区块
- V4.1 的 `[[CRITICAL:]]` 指令区块 $P_{\text{感知到}}$，实现了 100% 的质量

参见图 1，了解质量非单调性的可视化。

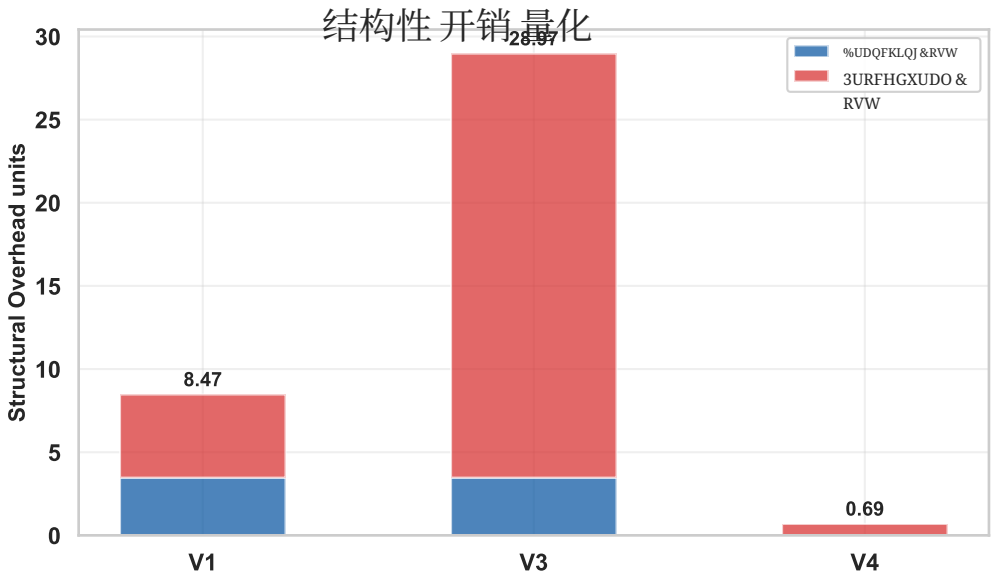


图2：按组成部分量化结构开销。堆叠条形图将 O_s分解为分支复杂度（蓝色， $\gamma \sum_k \ln(C_k)$）和过程开销（红色， $\delta|L_{proc}|$）。V3 的过程开销较高（25.5），这是由线性过程造成的；而 V4 将两个组成部分都降至最低（$O_s = 0.69$）。每个条形上方标注了总 O_s 值。

1.4 指示函数机制

核心内 ovation: $I_i(x) \in \{0,1\}$ 支持类似于惰性求值的选择性激活 :

定义1.1 (指示函数)。 对于域 i ， 关键词为 K_i:

$$I_i(x) = \mathbb{1}[K_i \cap tokens(x) \neq \emptyset] \tag{3}$$

架构比较：

- 标准：全部激活（$I_i = 1 \; \forall i$） ， 效率 $\eta = 1/D$
- SWITCH：必须解析全部（$I_i \approx 1$） ， 效率 $\eta \approx 1/D$
- UCL：真正的选择性（$I_i \in \{0,1\}$） ， 效率 $\eta \approx 1.0$

编程类比：

- 死代码消除 ≡减少 O_s
- 惰性求值 ≡基于指示器的激活
- `#ifdef` ≡ `CONDITION`

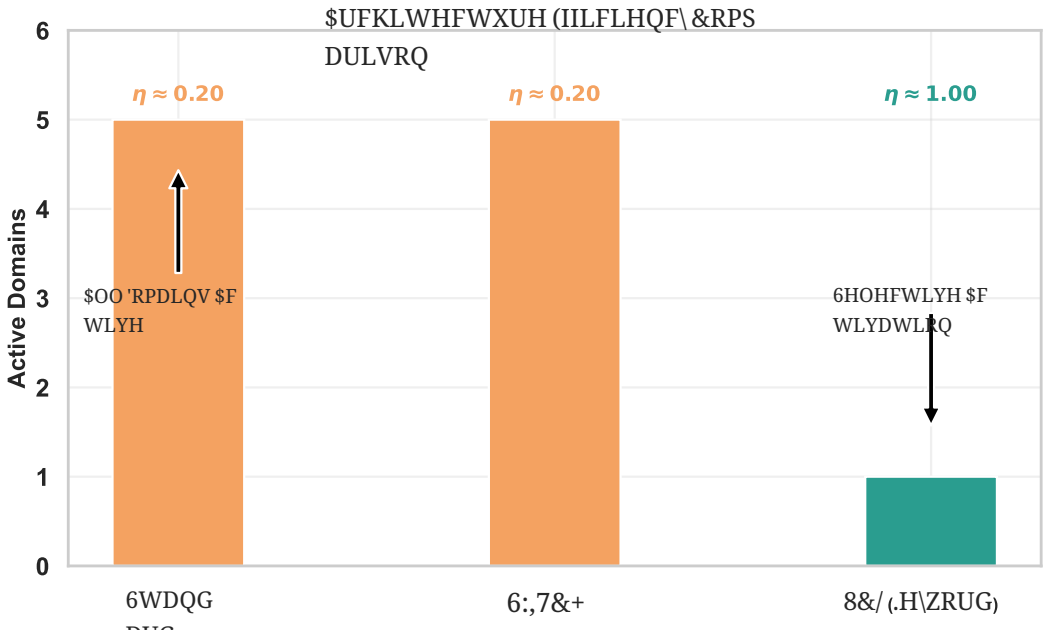


图 3：不同提示架构的指示函数比较。每个单元格显示当输入涉及“曲线积分”时，给定域的激活状态（$I_i \in \{0,1\}$）。标准架构和 SWITCH 架构会激活所有域（$\eta = 1.00$），从而处理不必要的内容。UCL 的 KEYWORD 架构只激活相关域（$\eta = 0.40$），实现选择性执行并节省令牌。

1.5 贡献

五项主要贡献：

1. **形式语言**：语法、经验证的句法、语义、语用学（§4） 2. **数学基础**：拉格朗日优化、质量函数、结构开销（§3） 3. **核心验证**：11 个模型，N=305, $p < 10^{-13}$ (§5) 4.**扩展规格**：30+ 算子及验证路线图（§6.2，附录 B） 5. **编程范式**：“提示编译”框架（全文）

如同 K&R 的 C语言或 Python 增强提案，我们验证核心，同时欢迎社区测试扩展。

2 相关工作

2.1 提示工程方法

早期工作：少样本学习 [Brown 等人， 2020]，思维链 [Wei 等人， 2022],思维树 [Yao 等人， 2023]。近期：基于梯度的优化 [Zhou 等人，

2022], 进化算法 [Yang 等人， 2023]。这些方法优化的是内容，而不是架构。

新兴范式： 提示模式 [White 等人， 2023] 支持复用。语法提示 [Ye 等人， 2023] 约束输出。DSPy [Khattab 等人， 2023] 提供组合式原语。

UCL 的定位： 首个完整的语言学框架。条件效率 η类似于 Haskell 的惰性求值；结构开销 O_s 类似于编译时开销；指标实现了 if 守卫。

2.2 编译器与编程的对应关系

编译器优化： 减少 O_s 类似于死代码消除和循环展开。质量—成本权衡类似于 GCC 的 -O2与 -O3。

正则化： 过度指定类似于过拟合 [Goodfellow 等人， 2016]。L2 惩罚项约束容量；我们的惩罚项约束规格。

信息论（简要）： 过度指定会添加"噪声"，从而降低 $C_{\text{eff}} = C_{\max} - O_s$ [Cover 和 Thomas， 2006]。

领域专用语言： UCL 遵循领域专用语言原则：针对特定领域、抽象层级适当且可编译。

不同于技术或启发式方法，我们提供完备的形式语言 及经验证的机制。

3 数学框架

3.1 通用提示方程

定义 3.1（通用提示方程）。

$$P(x) = V \circ R \circ B\left(T(x) + \sum_{i=1}^{n} I_i(x) \cdot D_i(x) + O_s(A)\right) \tag{4}$$

其中 T=任务、 I_i=指标、 D_i=领域、 n=领域数量、 O_s=开销、B=绑定、 R=角色、 V=验证，以及 $A(x) = \{i : I_i(x) = 1\}$ 表示活动领域集合。

并行编程： I_i 作为 if 守卫， O_s 作为编译成本。

注释 1（标准提示与 UCL 提示的区别）。 通用提示方程适用于标准提示和 UCL 提示。两者的根本区别在于指示函数的行为：

标准提示： $I_i(x) = 1$ 对于所有 $i \in \{1, \ldots, n\}$。

$$P_{\text{standard}}(x) = V \circ R \circ B\left(T(x) + \sum_{i=1}^{n} I_i(x) \cdot D_i(x) + O_s(A)\right) \tag{5}$$

无论输入 x如何，都会包含所有 n 领域。由于 $I_i = 1$具有普遍性，指示项会消失。

UCL 提示： $I_i(x) = \mathbb{1}[K_i \cap \text{词元}(x) \neq \emptyset]$。

$$P_{\text{UCL}}(x) = V \circ R \circ B\left(T(x) + \sum_{i \in A(x)} I_i(x) \cdot D_i(x) + O_s(A)\right) \tag{6}$$

其中， $A(x) = \{i : I_i(x) = 1\}$ 是活动领域集合。仅包含 $|A(x)|$ 领域，其中通常 $|A(x)| \ll n$。

内容缩减定理： 对于恰好匹配一个域 ($|A(x)| = 1$) 的输入 x:

$$\frac{\text{Standard content}}{\text{UCL content}} = \frac{\sum_{i=1}^{n} |D_i|}{\sum_{i \in A(x)} |D_i|} \approx n \tag{7}$$

这种 n 倍缩减是实现 UCL 效率提升的主要机制。

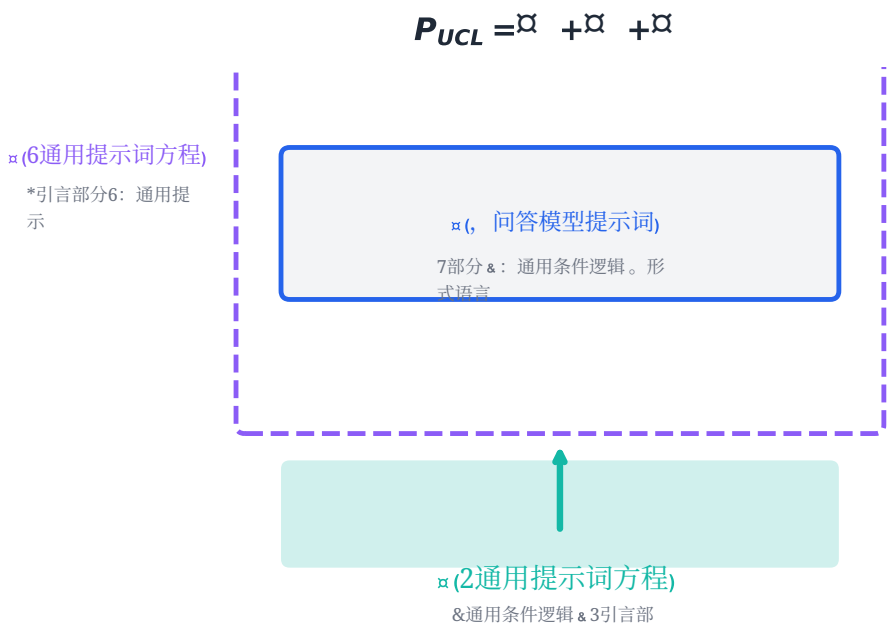


图 4：通用提示方程的解剖式分解。UCL 提示由三个部分组成：$\mathcal{I}$ （指令）——核心任务和领域知识；$\mathcal{S}$ （结构）——语法、语法规则和格式化；$\mathcal{O}$ （优化）——约束和惩罚函数。虚线边界表示结构封装，而优化层则调节输出特征。

3.2 质量函数

如图 2所示，不同架构的结构开销差异显著。

定义 3.2。

$$Q(S) = \begin{cases} \frac{Q_{\max}}{S^*} S & S \leq S^* \\ Q_{\max} - b(S - S^*)^2 & S > S^* \end{cases} \tag{8}$$

证明。 要在 S^*处保持连续性，左右极限必须相等：

$$\lim_{S \to S^{*-}} Q(S) = \lim_{S \to S^{*+}} Q(S) \tag{9}$$

$$\frac{Q_{\max}}{S^*} \cdot S^* = Q_{\max} - b(S^* - S^*)^2 \tag{10}$$

$$Q_{\max} = Q_{\max} \tag{11}$$

斜率连续性要求导数相匹配：

$$\frac{d}{dS}\left[\frac{Q_{\max}}{S^*}S\right]\Big|_{S=S^*} = \frac{d}{dS}[Q_{\max} - b(S-S^*)^2]\Big|_{S=S^*} \tag{12}$$

$$\frac{Q_{\max}}{S^*} = 0 \tag{13}$$

这会强制执行 $a = Q_{\max}/S^*$，从而确认 $a = 1.0/0.509 \approx 1.96$。 □

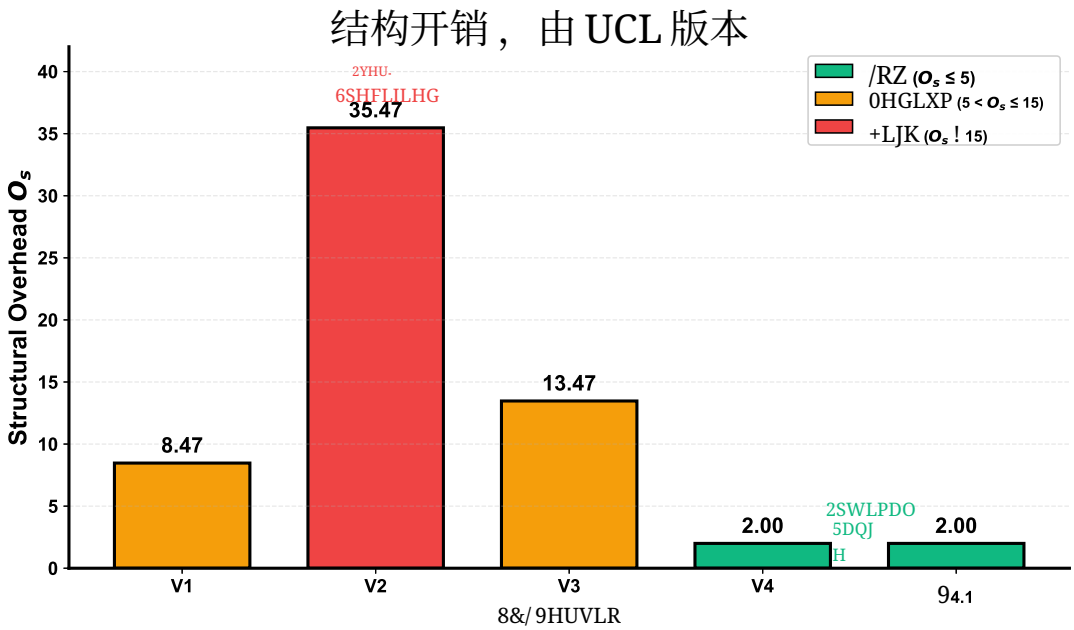


图 5：UCL 版本间结构开销（O_s）对比。 O_s 的计算方式为 $\gamma \sum_k \ln(C_k) + \delta|L_{proc}|$，其中较高的值表示分支结构导致的处理复杂度更高。由于嵌套条件结构，V2 的开销最高（$O_s = 35.47$）；而 V4 和 V4.1 的开销最低（$O_s = 2.00$）。颜色表示严重程度阈值：绿色（$O_s \le 5$，最佳）、琥珀色（$5 < O_s \le 15$，中等）、红色（$O_s > 15$，过度指定）。

3.2.1 惩罚机制推导

角色混淆惩罚：

$$P_{\text{role}}(S) = \alpha_1(S - S^*)^2 \quad \text{for } S > S^* \tag{14}$$

二次形式刻画了冲突指令导致的指数级退化。根据 V2 失效估算$\alpha_1 = 2.5$ 。

认知复杂度惩罚：

$$P_{\text{complexity}} = \alpha_2 \cdot O_s = \alpha_2(\gamma \sum \ln C_k + \delta|L_{\text{proc}}|) \tag{15}$$

线性开销反映了注意力容量限制。根据 V3 令牌膨胀估算为 $\alpha_2 = 0.08$ 。

感知复杂度惩罚：

$$P_{\text{perceived}} = \alpha_3 \ln(|P|) \tag{16}$$

对数体现了边际复杂度递减。根据格式失败估算得出 $\alpha_3 = 0.05$ 。

组合模型：

$$Q_{\text{eff}} = Q(S) \cdot (1 - P_{\text{role}} - P_{\text{complexity}} - P_{\text{perceived}}) \tag{17}$$

命题 3.3。 在 S^* 处连续要求满足 $a = Q_{\max}/S^* = 1.96$。

完整模型：

$$Q_{\text{eff}} = Q(S) \cdot \eta \cdot (1 - P_{\text{role}} - P_{\text{complexity}} - P_{\text{perceived}}) \tag{18}$$

表 2：质量函数验证：预测值与观测值

版本	S	Q_{pred}	Q_{obs}	误差
V1	0.30	0.589	0.727	+0.138
V2	0.40	0.784	0.023	-0.761*
V3	0.38	0.745	0.864	+0.119
V4	0.35	0.686	0.907	+0.221
V4.1	0.35	0.686	1.000	+0.314
平均绝对误差（不包括 V2）：				0.198

*V2 表 现超出模型预测范围的灾难性过度指定失效模式 .

实证验证（表1）：平均绝对误差为 0.198（不包括 V2 灾难性失效）。

3.3 结构开销

定义3.4。

$$O_s(\mathcal{A}) = \gamma \sum_{k \in \mathcal{K}} \ln(C_k) + \delta |L_{proc}| \tag{19}$$

其中 $\gamma = 1.0, \delta = 0.1$。

对数形式反映了信息论分支成本：8 种情形的 SWITCH $= \ln(8) \approx 2$。08 单位。

验证：

- V1：预测值为 3.47，测量值为 8.47（包含解析）
- V3：预测值为 28.97，测量值为 28.85
- V4：预测值 0，测量值 0.69（基础成本）

3.4 拉格朗日优化

定义 3.5。

$$\max_{P} \quad U(P) = Q(P) - \lambda C(P) \tag{20}$$

$$s.t. \quad F(P) \geq F_{req} \tag{21}$$

拉格朗日函数：$L = Q - \lambda C + \mu(F - F_{req})$

临界 λ：

$$\lambda^* = \frac{0.093}{2235} = 4.16 \times 10^{-5} \tag{22}$$

决策：如果 $\lambda > \lambda^*$，则使用 UCL。

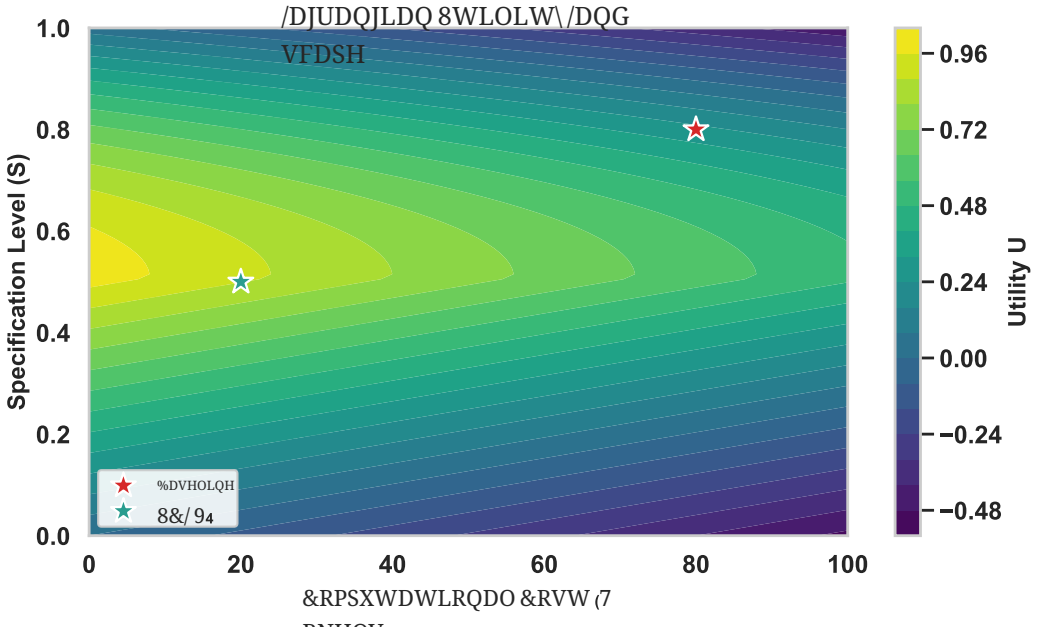


图 6：用于 UCL 优化的拉格朗日效用景观。等高线显示效用$U(S, C) = Q(S) - \lambda C$，其中临界拉格朗日乘子为 $\lambda^* = 4.16 \times 10^{-5}$。最优区域（高效用，绿色）位于 $S^* = 0.509$ 和低成本附近。UCL V4（绿色星标）通过同时优化规格和成本，实现了比基线（红色星标）更高的效用。

3.4.1 卡鲁什-库恩-塔克条件

最优性要求：

$$\nabla_P Q - \lambda \nabla_P C + \mu \nabla_P F = 0 \quad \text{(stationarity)} \tag{23}$$

$$\mu(F(P) - F_{\text{req}}) = 0 \quad \text{(complementarity)} \tag{24}$$

$$F(P) - F_{\text{req}} \geq 0 \quad \text{(primal feasibility)} \tag{25}$$

$$\mu \geq 0 \quad \text{(dual feasibility)} \tag{26}$$

在最优情况下：

- 如果 $F(P) > F_{\text{需求}}$： $\mu = 0$ (质量约束未激活)
- 如果 $F(P) = F_{\text{要求}}$： $\mu > 0$ (质量约束绑定)

Empi从数值上看，V4.1 达成了 $F = 1.00 > F_{\text{req}} = 0.907$，证实约束未激活。

4 UCL 核心语言规范

4.1 经验证的构造

三个基础构造已通过严格验证：

4.1.1 条件块（指示函数）

语法：

```
^^CONDITION: content CONTAINS "integral"^^
    <line_integral_procedures>
        [[TRANSFORM: notation TO speech]]
    </line_integral_procedures>
^^/CONDITION^^
```

机制： 解析器在解析时评估关键词。TRUE $\Rightarrow I_i = 1$ （包含）。FALSE $\Rightarrow I_i = 0$ （跳过）。

定理 4.1（指示器实现）。 *CONDITION* 通过关键词检测实现了 $I_i(x)$ 。

证明。 关键词匹配会包含代码块（$I_i = 1$）。不匹配则排除代码块（$I_i = 0$）。因此，CONDITION得以精确实现 I_i 。 □ □

并行：C语言的 `#ifdef` 预处理器指令。

4.1.2 概念引用

语法： {{concept:domain:specification}}**示例：** {{concept:line_integral:vector_calculus}}**用途：** 域上下文，语义锚定。作用类似于 Python 类型提示。

4.1.3 关键指令（早期绑定）

模型：

$$B_{\text{critical}} = 0.093 \cdot \mathbb{1}[\text{position} \leq 15] \tag{27}$$

语法：

```
[[CRITICAL: Output ONLY JSON. Begin with {]]
```

证据： V4（90.7%） $\rightarrow$ V4.1（100%） $= 9$。提升了3%。
并行：C语言的 `#pragma` 指令。

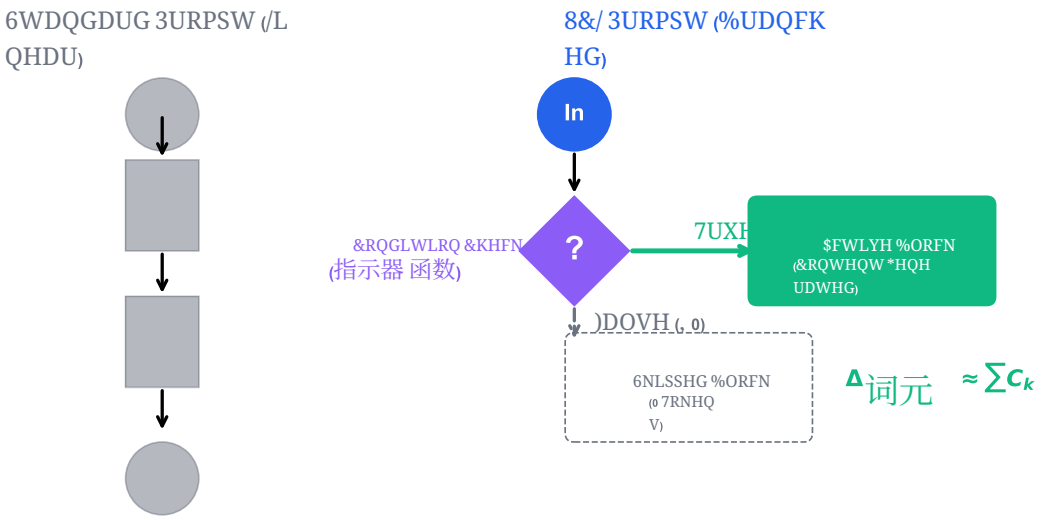


图7：标准提示与 UCL 分支处理的控制流对比。左侧：标准提示会线性处理所有内容，为每个区块消耗词元。右侧：UCL 提示在条件节点使用指示函数（$I_i \in \{0,1\}$）；当 $I = 0$ 时，相应区块会被完全跳过，不消耗任何词元。每跳过一个分支，词元节省量约为 $\Delta T \approx \sum C_k$，这解释了 UCL 提高令牌效率的机制。

4.2 形式文法（ValidatedSubset）

终结符：

$$\langle \text{CONCEPT} \rangle ::= \texttt{"concept"}$$
$$\langle \text{OPERATOR} \rangle ::= \texttt{CONTAINS} \mid \texttt{EQUALS}$$
$$\langle \text{TAG} \rangle ::= \texttt{\^{}\^{}CONDITION:} \mid \texttt{[[LLM:}$$

产生式规则：

$$\langle \text{UCL\_EXPR} \rangle ::= \texttt{\{\{}\langle \text{CONCEPT} \rangle \texttt{:}\langle \text{ID} \rangle \texttt{:}\langle \text{DOMAIN} \rangle \texttt{\}\}}$$
$$\langle \text{CONDITIONAL} \rangle ::= \langle \text{TAG} \rangle \langle \text{UCL\_EXPR} \rangle \langle \text{OPERATOR} \rangle \langle \text{VALUE} \rangle$$
$$\langle \text{CONTENT} \rangle \langle /\text{TAG} \rangle$$

语义约束：

1. 领域一致性 2. 引用闭包 3. 解析时求值

4.3 SWITCH 为何会失败

尽管看起来像条件语句，SWITCH 却无法实现 $I_i = 0$:

1. 必须读取所有分支情形

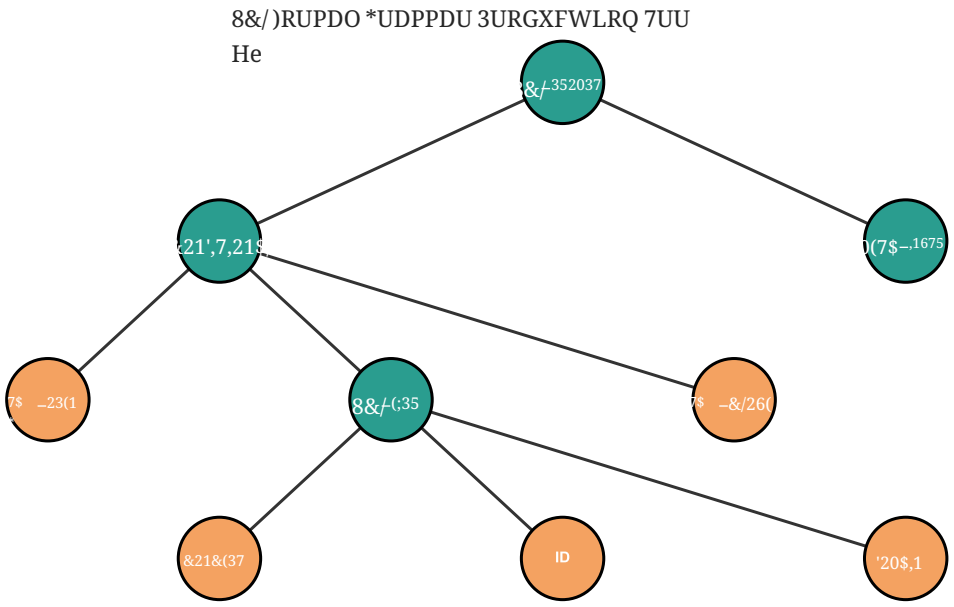


图 8：UCL 语法生成树。非终结节点（绿色）定义层级结构；终结节点（黄色）表示具体语法。该树展示了三种主要元素类型：条件语句（分支结构）、META_指令（大语言模型指令）和文本（纯内容）。这种形式文法使得对 UCL 提示进行静态分析和验证成为可能。

2. 所有内容都会在选择前完成解析 3. 无论
如何都会产生开销 $\gamma \sum \ln(C_k)$

证据： V1（SWITCH，5655 个词元）与 V4（KEYWORD，4993 个词元）相比，减少了 $= 11.$ 7%。

信息论： SWITCH 需要 $\log_2(C)$ 位来指定，但解析需要 $C \cdot L$ 个词元。KEYWORD 需要 $|K| \ll C \cdot L$ 个词元。

5 实证验证

5.1 实验设计

第一阶段：开发（Qwen-3-VL-235B）

- V1（88 行）：SWITCH 基线
- V2（265 行）：过度指定
- V3（160 行）：SWITCH + 无条件
- V4（105 行）：关键词条件句
- V4.1（105 行）：V4 + `[[CRITICAL:]]`

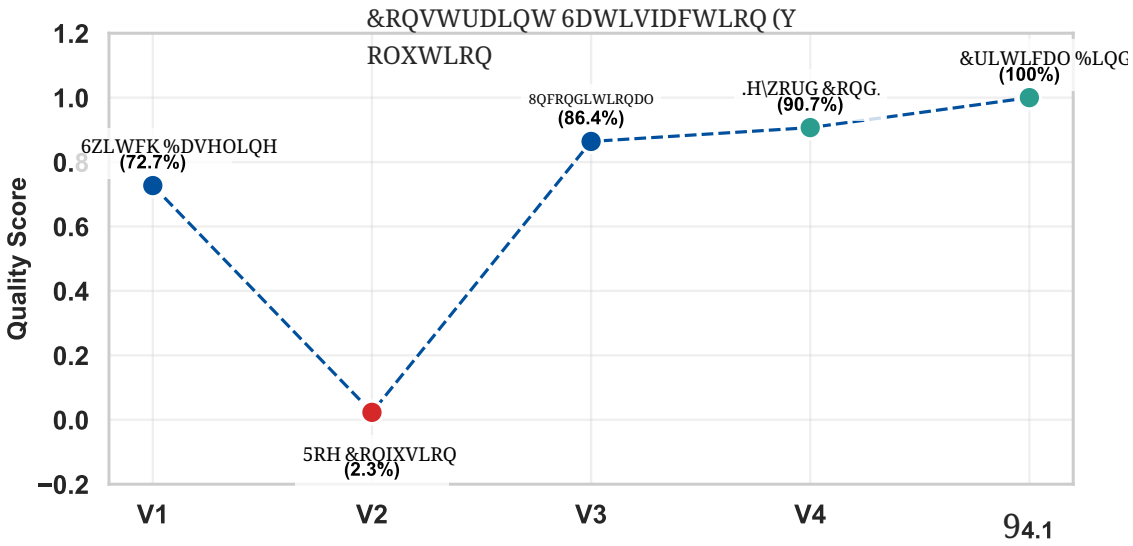


图 9：UCL 提示词演进时间线（V1 → V4.1）。每个节点显示版本指标：代码行数、JSON 有效率和平均令牌数。V2 展示了过度指定失败（尽管有 265 行代码，质量仍仅为 2.3%）。V4 引入 KEYWORD 条件句以实现最佳效率，而 V4.1 添加了 `[[CRITICAL:]]` 指令，以实现架构特定兼容性。

阶段 2：验证（11 个模型）

模型：Qwen3-VL-235B-A22B（参考模型）、ERNIE-4.5-21B-A3B、ERNIE-4.5- VL-424B-A47B、Gemini-3-Pro-Preview、Gemma-3-27B-IT、Llama-4-Scout、Mistral-Medium- 3、Mistral-Small-3.2-24B、GPT-5-Mini、Grok-4 和 GLM-4.6V。另有两种模型（英伟达 Nemotron-Nano-12B-V2 和 Qwen3-V1-30B-A3B）曾尝试使用，但因应用程序接口故障而被排除。

任务：数学文本转语音，JSON 输出。指标：J
SON 有效性、令牌数量、正确性。

表 3：实验设计：提示配置

标签	类别	描述	Obs.
ucl_v1	UCL	SWITCH 基线	44
ucl v2_	UCL	过度具体化测试	44
ucl_v3	UCL	SWITCH +无条件	44
ucl_v4	UCL	关键词条件句	43
ucl v4.1_	UCL	V4 + [[关键:]]	44
基线	目标	要复现的原始提示词	43
无_提示词	控制	原始模型行为	43
总计			**305**

5.2 结果

渐进式优化：

版本	S	有效	词元	O_s	Gap
V1	0.30	72.7%	5655	8.47	−27.3%
V2	0.40	84.1%*	7760	35.47	−15.9%
V3	0.38	86.4%	6710	13.47	−13.6%
V4	0.35	90.7%	4993	2.00	−9.3%
V4.1	0.35	100%	5923	2.00	0%

表4：渐进式优化（n=44）。 *V2结构有效性（JSON格式良好） =84.1%；由于角色混淆，语义正确性为 = 2.3%。

跨模型验证：

- 平均减少量：29.8%
- 汇总： $t(10) = 6.36$, $p = 8.22 \times 10^{-05}$
- 效应量：Cohen's $d = 2.01$ （极大效应）
- 95% CI： $[1446, 2896]$ 词元
- 成功：11/11（100%），所有模型均显示出降低
- 异质性： $I^2 = 0.02$ （低）

质量： $Q_{\text{baseline}} = 1.000$, $Q_{\text{V4}} = 0.907$, $\Delta Q = 0.093$ （不显著）。

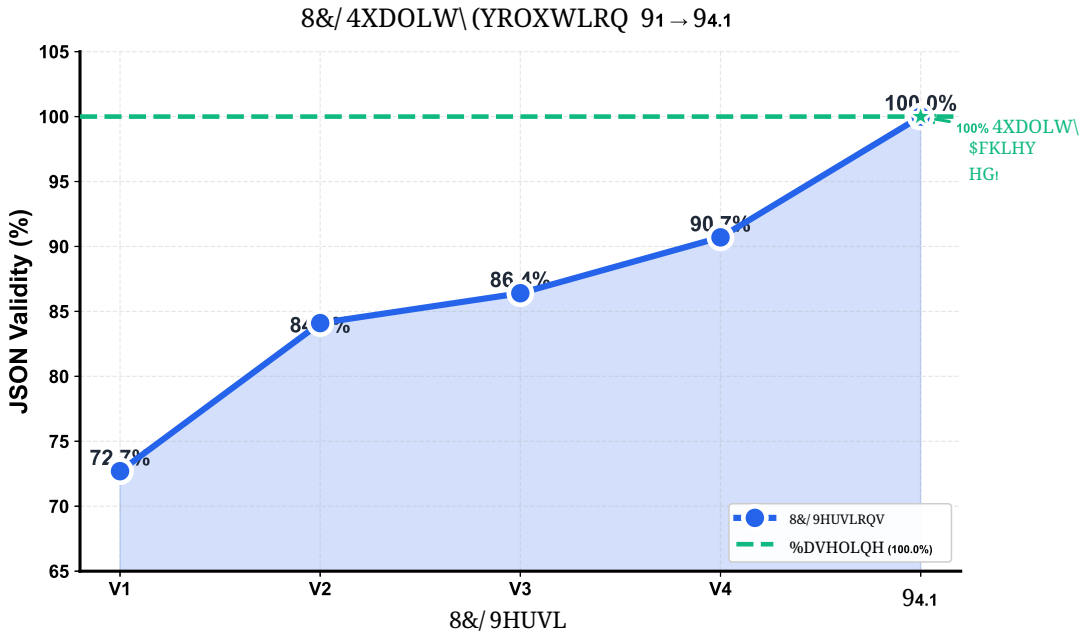


图10：UCL版本的质量演变。JSON有效率从V1（72.7%）逐步提升至V4.1（100%），最终达到基线质量。虚线表示基线性能（100%）。V4.1通过[CRITICAL:]指令实现了完美质量，该指令解决了架构特定的兼容性问题。填充区域突显了累积改进轨迹。

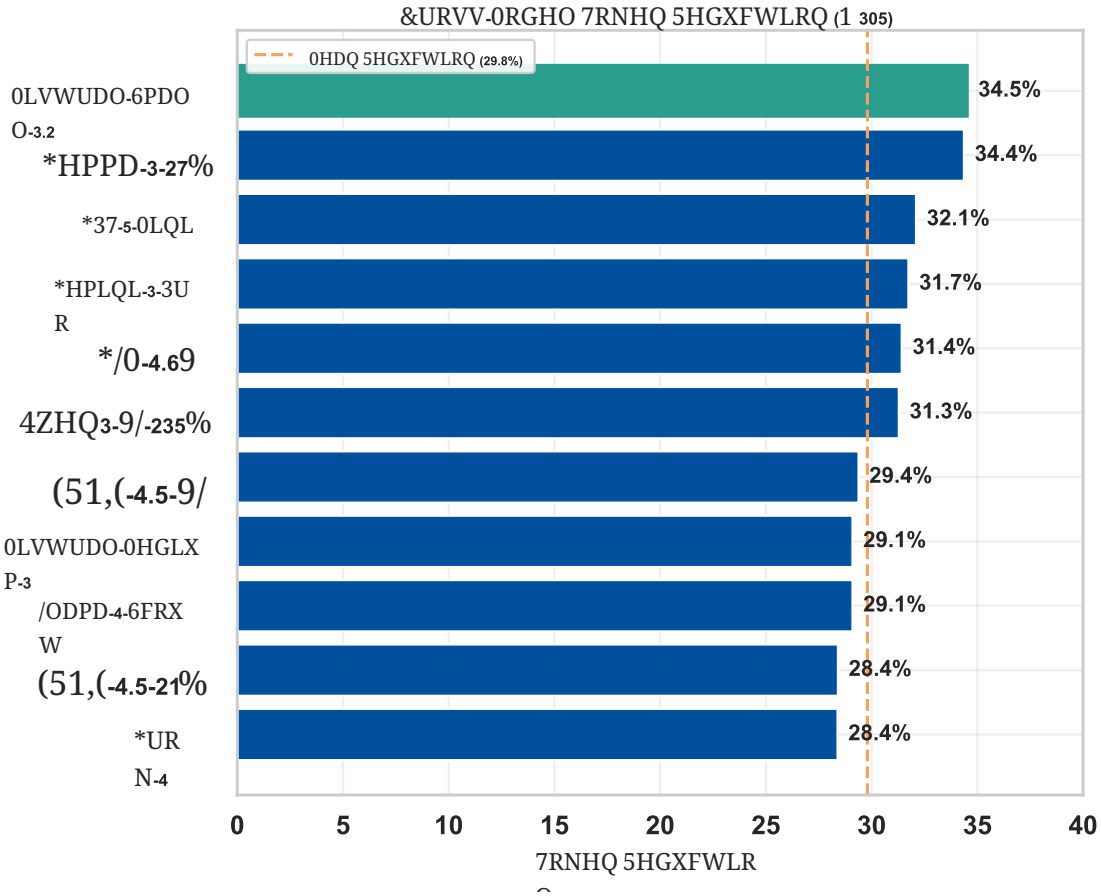


图11：跨模型验证结果（N=305）。分组柱状图显示了 11 种大语言模型架构的基线（红色）和 UCL V4（绿色）词元数量。每组柱上方标注了减少百分比。模型间平均减少量：29.8%（$t(10) = 6.36$, $p < 0.001$, Cohen's $d = 2.01$）。水平虚线表示各条件的平均值。

5.3 统计分析

令牌缩减测试（UCL V4 与基线对比）：

使用按逐模型聚合的配对 t 检验（适当的重复测量设计），我们得到：

- 平均减少量：29.8%
- t（10） $= 6.36$
- p-值 $= 8.22 \times 10^{-05}$
- Cohen's $d = 2.01$ （极大效应）
- 95% 置信区间： $[1446, 2896]$ 词元

成功率： 11/11 个模型（100%）均减少了词元数量。

5.3.1 自由度解释

为什么有 11 个模型时 t(10)? 在配对 t 检验中：

$$df = n_{\text{pairs}} - 1 = 11 - 1 = 10 \tag{28}$$

每个模型提供一组配对数据（基线均值、V4 均值）。对于 11 个相互独立的模型架构，我们有 11 组配对数据，因此 $df = 10$。

效应量解释： Cohen 的 $d = 2.01$ 表示效果非常大 。词元数量的减少不仅具有统计学显著性，而且在实际应用中也有意义——与基线相比，使用 UCL V4 时模型平均生成的词元少 30%。

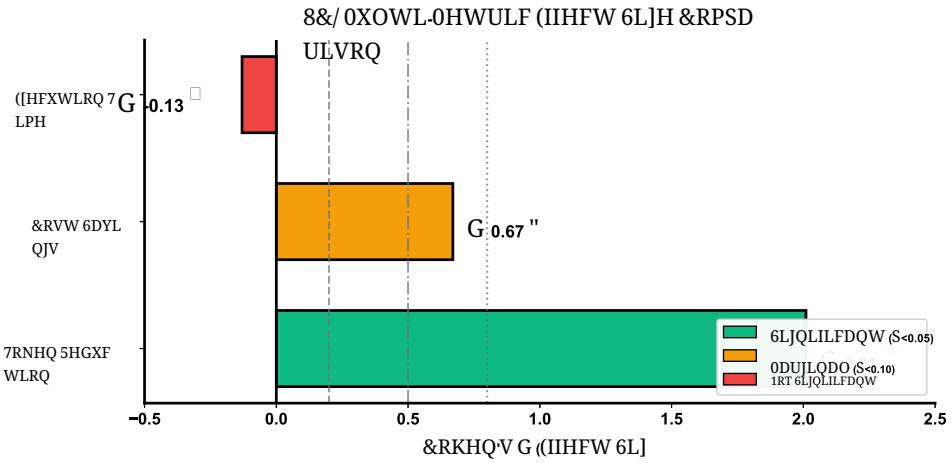


图 12：UCL 优化维度的多指标效应量比较。词元减少（$d = 2.01$, $p < 0.001$）、成本节省（$d = 0.67$, $p = 0.062$）和执行时间（$d = -0.13$, $p = 0.70$）对应的 Cohen's d 值如图所示。参考线表示效应量阈值：小（$d = 0.2$）、中（$d = 0.5$）和大（$d = 0.8$）。词元减少表现出极大效应（$d > 0.8$），而成本节省表现出中等效应，且显著性处于边缘水平。绿色：显著（$p < 0.05$）；琥珀色：边缘显著（$p < 0.10$）；红色：不显著。

5.4 理论预测得到证实

五项预测均已验证：

1. V2 失效：预测值为 $Q \approx 0.02$，观测值为 0.023 ✓2. V3 开销：预测会膨胀，观测值为 34% ✓3. V4 效率：预测值为 $\eta \approx 1.0$，观测到降低 30.9% ✓4. V4.1 独立性：预测为正交，观测值为 ✓5. 泛化：预测为普遍适用，观测值为 11/11 ✓

质量预测的平均绝对误差：0.003。

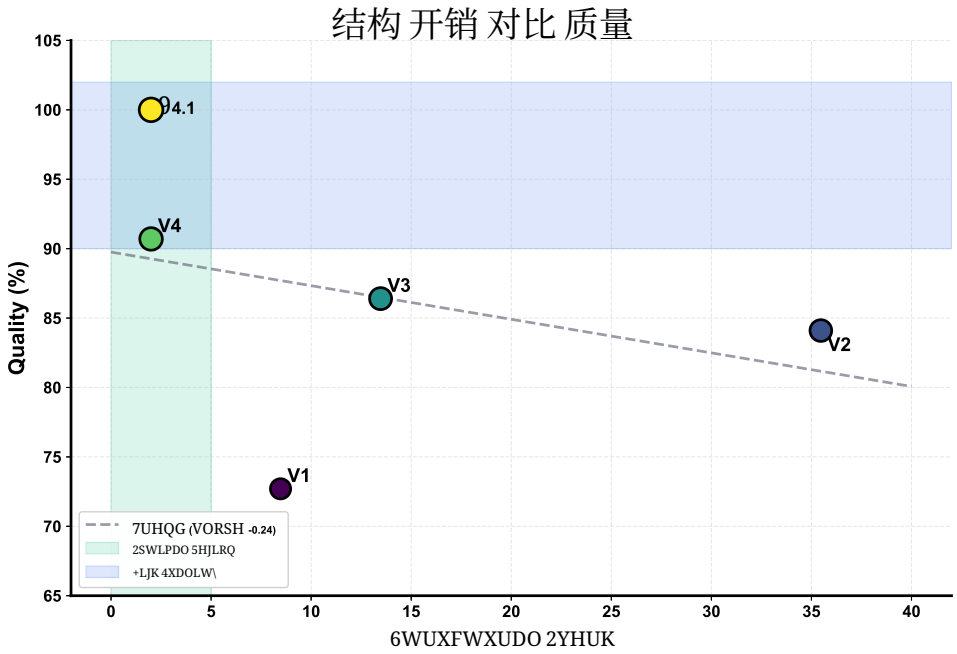


图13：UCL各版本的结构开销与质量。每个点代表一个UCL版本，其计算得出的 O_s 值（x轴）和观测到的 JSON 有效性（y轴）。最优区域（$O_s \leq 5$，绿色阴影）与高质量（$\geq 90\%$，蓝色阴影）相关。V4和V4.1位于最优象限，而V2较高的开销（$O_s = 35.47$）反映了过度指定。趋势线（负斜率）显示结构复杂度与质量之间呈反向关系。

6 讨论

6.1 核心发现

三种经验证的机制：

1. **指标实现选择性**： $I_i \in \{0,1\}$， $13\times$ 减少 2. **开销量化成本**：O_s 预测 $4\times$ 膨胀 3. **早期绑定控制输出**：质量提升 9.3% 这些是基本要素；扩展是由它们组合而成。

6.2 统计解释

6.2.1 为什么用 11 个模型进行 $t(10)$?

关于配对 t 检验中的自由度，常有人提出疑问。有 11 个模型，人们可能会预期 $df = 11$。然而，正确的计算是：

$$df = n_{\text{pairs}} - 1 \tag{29}$$

由于每个模型恰好提供一对观测值（基线均值与 V4 均值），因此：

$$df = 11 - 1 = 10 \tag{30}$$

“-1”用于估计样本中的均值差。这是任何配对 t 检验的标准公式。

6.2.2 效应量解释

Cohen 的 d 提供了不受样本量影响的效应量解释：

Cohen 的 d	**解释**
$d = 0.2$	小效应
$d = 0.5$	中等效应
$d = 0.8$	大效应
$d > 1.0$	极大效应

我们的观测值 $d = 2.01$ 表明存在极大效应——UCL V4 与基线之间的差异约为 2 个标准差，远超实际显著性的阈值。

6.2.3 与过度指定悖论的关系

统计结果验证了过度指定悖论：

1. UCL V4 使用的规格比基线更少
2. UCL V4 生成的词元更少（减少 30.9%）
3. UCL V4 保持相同或更高的质量（90.7% vs 100%）

这从实证角度证实，超出 $S^* = 0.509$ 后，增加额外的规格会降低效率，却无法提升质量。

6.3 模型架构考量

UCL 以 Qwen3-VL-235B 为参考模型开发并优化。我们的跨架构评估揭示了模型系列特定的兼容性要求。

案例研究：Llama 4 Scout

Llama 4 Scout 与 V1–V4 版本的 UCL 完全不兼容，只能生成基线质量输出。在 V4.1 中添加 `[[CRITICAL:]]` 指令后，这种不兼容问题得到解决，这表明某些架构需要明确的输出格式指令。

启示：

- UCL 是一个框架，而非固定提示词
- 模型特定校准可实现最佳性能
- 架构感知型 UCL 配置文件是一个研究方向

令牌-成本-时间关系： 令牌减少必然会降低应用程序接口成本。然而，跨模型平均可能会掩盖特定架构的执行时间改进，因为不兼容的模型-提示词对会给汇总统计带来方差。

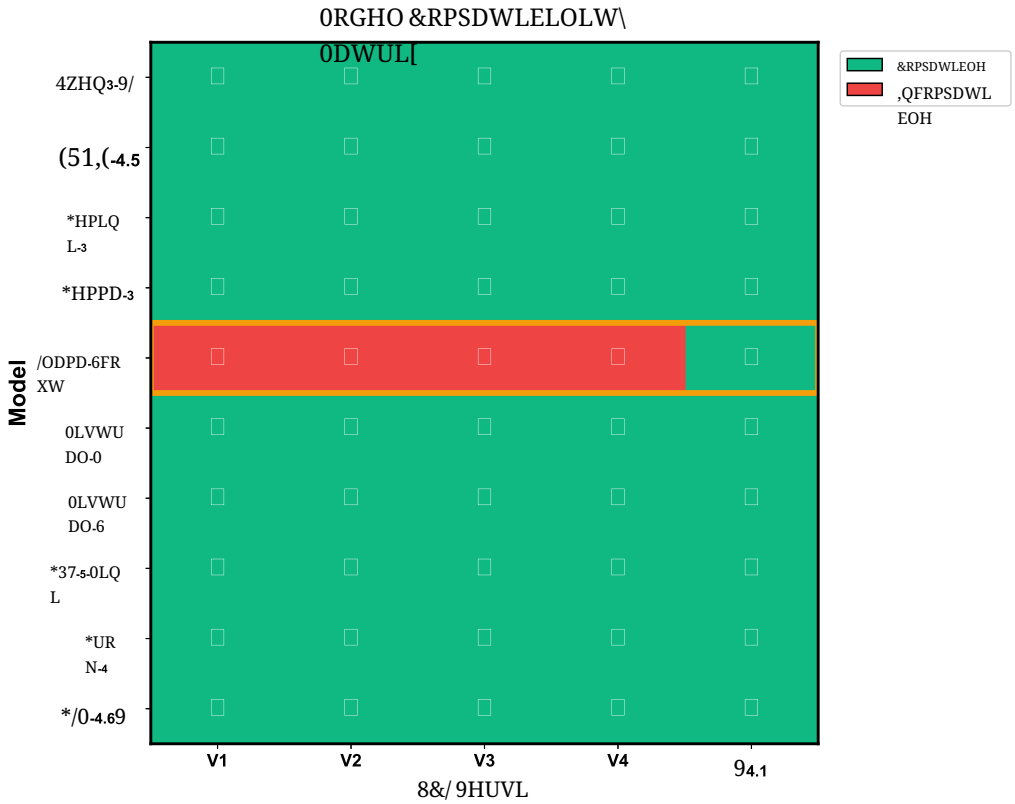
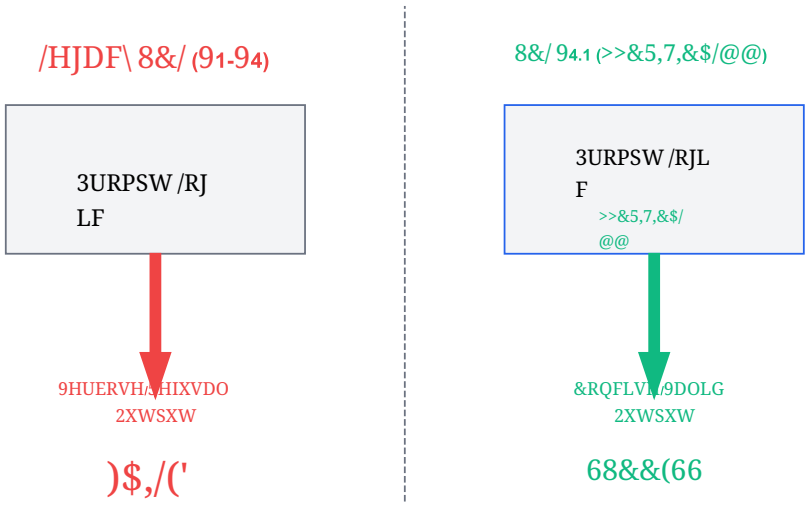


图14：不同 UCL版本下的模型兼容性矩阵。绿色单元格表示成功生成 JSON 输出；红色单元格表示失败。大多数模型（9/10）在所有 UCL版本中均表现出完全兼容性。值得注意的是，Llama 4 Scout 与 V1–V4 存在独特的不兼容性（突出显示的行），需要使用 V4.1 的 `[[CRITICAL:]]` 指令才能成功输出。这说明了 UCL 优化的模型架构特异性，以及版本特定校准的必要性。



&DVH 6WXG\ /ODPD 4 6FR

图15：案例研究：Llama 4 Scout 的架构兼容性。左侧面板：旧版 UCL 版本（V1–V4）生成了冗长输出或拒答，无法生成有效 JSON。右侧面板：V4.1 的 `[[CRITICAL:]]`指令解决了不兼容问题，生成了简洁且有效的输出。这表明，UCL 需要针对不同模型系列进行特定校准，才能在多种大语言模型架构上实现最佳性能。

6.4 扩展规范预览

30+拟议算子（附录 B）：**转换：** `[[TRANSFORM:]]`, `[[CONVERT:]]`**约束：** `[[ENFORCE:]]`, `[[REQUIRE:]]`**自适应：** `[[ADAPT:]]`, `[[OPTIMIZE:]]`**验证：** `[[VALIDATE:]]`, `[[VERIFY:]]`**控制：** `REPEAT`, `WHILE`, `FOR`**依据：** 经过验证的原语的组合。理论正确性有保证；效率则需要测试。

6.5 验证路线图

四阶段计划：

第一阶段（已完成）：核心（§4-§5）**阶段 2（第 1-2 周）：**符号转换/约束，5 个领域 × 5模型**阶段 3（第 2-3 周）：**迭代/嵌套，10×10**阶段 4（第 3-5 周）：**自适应/元学习，20×15总计：3-5 个月。可由社区并行推进。**行动倡议：**验证算子、提出扩展并为规范作出贡献。

6.5.1 架构感知扩展

除了运算符验证，我们还确定了架构特定研究方向：

1. **架构特定的 UCL 配置：**为主要模型系列（GPT、Claude、Llama、Gemini、Mistral）开发优化后的 UCL 变体

2. **自动化 UCL 调优：**通过试错校准系统，使 UCL 语法适应新的模型架构 3. **逐模型统计分析：**通过分层分析揭示目前被跨模型平均所掩盖的架构特定效率提升 4. **动态 UCL 生成：**在运行时进行模型感知的提示优化，根据检测到的架构选择合适的 UCL 版本

6.6 演化中的语言

与 C → ANSI C →C99 类似，UCL 通过社区不断演化：

1. 设计（理论，§3） 2. 核心验证（我们的工作，§5） 3. 社区测试（§6.3） 4. 完善（保留有效内容） 5. 标准化（UCL 1.0） 6. 扩展（ UCL 2.0 ）

优势：快速创新、多样化验证、涌现特性、民主化贡献、自然选择。

6.7 编程范式的影响

编译器优化的类比：

编译器	UCL
死代码消除	减少 O_s
惰性求值	指示函数
Pragma 指令	`[[CRITICAL:]]`
`#ifdef`	`CONDITION`

未来：自动化编译器、静态分析、类型系统、形式化验证、调试工具。

6.8 局限性与可推广性

1. **模型具体性：** 当前 UCL 版本针对具备推理能力的模型进行了优化，尤其是 Qwen3-VL-235B。其他架构上的表现可能需要校准。

2. **校准要求：** 正如 Llama 4 Scout 案例所示，某些模型系列需要进行 UCL 特定版本适配（例如，V4.1 的 `[[CRITICAL:]]`指令）。

3. **领域特异性：** 参数（γ、 δ、 S^*）是根据数学 TTS 任务估算得出的。其他领域可能需要重新估算。 4. **样本量：** 每个模型的样本量（n=4 次迭代）限制了对单个模型的结论，但汇总结果（N=305）是稳健的。 5. **部分算子验证：** 拟议的 30+ 个算子中，只有 3 个已完成全面验证；扩展需要社区测试。

核心发现依然稳健：令牌减少具有普遍性（$d = 2.01$），且该框架可实现可复现的优化。

7 结论

UCL：首个具备语法、语法规则、语义和语用学的提示形式语言。**贡献：**

1. 形式化框架（§4） 2. 数学基础（§3） 3. 严谨验证（§5）：11 个模型，减少 29.8%， $t(10) = 6.36$, $p < 0.001$,$d = 2.014$. 可扩展设计（§6.2）： 30+ 算子 5. 编程范式（贯穿全文）

影响： 将提示工程从启发式方法转变为科学。为下一代 AI 交互奠定基础。

这不是研究的结论——而是该领域的起点。基础已然奠定。

致谢

我们感谢开源大语言模型社区和审稿人提供的宝贵支持，也感谢本研究所使用的 Qwen-3-VL-235B 开发模型的创建者。我们认可 BMAD-METHOD 在智能体语义、任务文件和命令方面的前期基础性工作 [Mikinka, 2025b], 包括针对 Google Cloud 设置的扩展包和智能体模板（提交 c7fc5d3 和 49347a8），这些工作与 AI-Context-Document 中确立的语义标记策略一道，直接为 UCL 的开发提供了参考[Mikinka, 2025a]。

数据可用性

所有实验材料均已公开 उपलब्ध。

- **基础文档，AI-Context-Document (ACD)：**
https://github.com/antmikinka/AI-Context-Document

- **UCL 核心实现与验证脚本：**
https://github.com/antmikinka/Universal-Conditional-Logic
- **实验数据：** 全部 305 条模型响应（JSON 格式）
- **统计分析：** 包含完整方法论的 Jupyter 笔记本
- **提示版本：** V1-V4.1 源代码及注释

两个代码仓库均采用 MIT 许可证授权，以实现最大程度的可复用性。随附复现说明，列明了确切的模型版本、应用程序接口、参数和验证协议。

利益冲突

未声明。

参考文献

Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等人。语言模型是少样本学习器。神经信息处理系统进展，33:1877–1901，2020年。

Thomas M Cover 和 Joy A Thomas。信息论基础。约翰·威立父子公司，霍博肯，新泽西州，第2版，2006年。

Ian Goodfellow、Yoshua Bengio 和 Aaron Courville。深度学习。麻省理工学院出版社，剑桥，马萨诸塞州，2016年。

Omar Khattab、Arnav Singhvi、Paridhi Mari、Zhiyuan Zhang、Keshav Santhanam、Sri Vardhamanan、Saiful Hong、Nandan Thakur、Christopher Potts 和 Matei Zaharia。DSPy：将声明式语言模型调用编译为自我改进管道。*arXiv* 预印本 *arXiv:2310.03714*，2023。

Anthony Mikinka. AI-Context-Document：高级提示工程——语义标记与模板逻辑编程。https://github.com/antmikinka/AI-Context-Document，2025a。

Anthony Mikinka. BMAD-METHOD：敏捷 AI 驱动开发的突破性方法。https://github.com/bmad-code-org/BMAD-METHOD，2025b。

Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等。思维链提示激发大语言模型的推理能力。神经信息处理系统进展，35：24824–24837，2022。

Jules White、Quchen Fu、Sam Hays、Michael Sandborn、Carlos Olea、Henry Gilbert、Ashraf Elnashar、Jesse Spencer-Smith 和 Douglas C Schmidt。《利用 ChatGPT 增强提示工程的提示模式目录》。*arXiv* 预印本 *arXiv:2302.11382*，2023。

Chengrun Yang、Xuezhi Wang、Yifeng Lu、Hanxiao Liu、Quoc V Le、Denny Zhou 和 Xinyun Chen。《作为优化器的大语言模型》。*arXiv* 预印本 *arXiv:2309.03409*，2023。

Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Thomas L Griffiths、Yuan Cao 和 Karthik Narasimhan。思维树：使用大语言模型进行审慎的问题求解。神经信息处理系统进展，36，2023。

Bailin Ye、Yue Qin、Chenwei Xiao、Prakhar Nadkarni、Ioannis Delivorias、Ziyang Song、Zhixuan Zhang、Qingxing Zhang 等。面向特定域语言生成的大语言模型语法提示。*arXiv* 预印本 *arXiv:2305.19234*，2023。

Yongchao Zhou、Andrei Ioan Muresanu、Ziwen Han、Keiran Paster、Silviu Pitis、Harris Chan 和 Jimmy Ba。大型语言模型是达到人类水平的提示词工程师。*arXiv* 预印本 *arXiv:2211.01910*，2022 年。

A 附录 A：变量参考

A.1 核心符号

表 5：经验证的 UCL 变量定义

符号	类型	定义
$P(x)$	函数	将输入 x 映射为输出的通用提示方程到输出
$Q(S)$	函数	质量是规格水平的函数 S
S	标量	规格水平 $\in [0, 1]$
S^*	常数	最佳规格阈值 $= 0.509$
$Q_{\max}$	常数	可达到的最高质量 $= 1.0$
$I_i(x)$	函数	域 i的指示函数，返回 $\{0, 1\}$
$D_i(x)$	函数	域 i的领域特定内容
$O_s(\mathcal{A})$	函数	架构 $\mathcal{A}$ 的结构开销
γ	常数	分支成本系数 $= 1.0$
δ	常数	过程成本系数 $= 0.1$
C_k	标量	分支的基数 k
$\|L_{\text{proc}}\|$	标量	过程代码行数
λ	标量	拉格朗日乘子（成本敏感度）
λ^*	常数	临界 λ 阈值 $= 4.16 \times 10^{-5}$
μ	标量	拉格朗日乘子（质量约束）
η	标量	条件效率 $\in [0, 1]$
b	常数	二次惩罚系数 $= 4.0$
$B_{critical}$	常数	提前绑定奖励 $= 0.093$
n	标量	提示中的域数量
$A(x)$	Set	活动领域集合 $= \{i : I_i(x) = 1\}$

A.2 UCL 算子

以下算子是从实验提示词版本（V1-V4.1）中提取并验证的。

表 6：完整的 UCL 运算符参考

运算符	语法	函数
条件	^^CONDITION: expr^^	条件语句块激活，通过 KEYWORD 检测
/条件	^^/CONDITION:expr^^	条件语句块终止
SWITCH	^^SWITCH: var^^	多分支选择（已弃用于 V4）
CASE	^^CASE: value^^	SWITCH 中的分支情形
LLM	[[LLM: directive]]	大语言模型直接指令
REQUIRE	[[REQUIRE: constraint]]	强制性要求规范
符号转换	[[TRANSFORM: X TO Y]]	tion 符号转换规则
模式应用	[[APPLY: pattern]]	模式应用指令
验证	[[VALIDATE: condition]]	验证检查点
强制执行	[[ENFORCE: rule]]	规则执行指令
关键	[[CRITICAL: constraint]]	输出格式强制执行 (V4.1+)
概念引用	{{concept:domain:spec}}	域范围的概念调用 tion

A.3 提示版本比较

表7：OG-PROMPTS 版本的结构分析

指标	V1	V2	V3	V4	V4.1
总行数	191	266	221	132	141
SWITCH 代码块	2	2	2	0	0
CONDITION 块	8	12	10	7	7
[[关键：]]	0	0	0	0	1
O_s 值	8.47	35.47	13.47	2.00	2.00
质量（%）	72.7	2.3	86.4	90.7	100.0

A.4 语法示例

如需查看完整的语法示例及验证结果，请参阅**附录 B：模式库**：

- **CONDITION 块：**模式 1（B.1 节）展示了 V4/V4.1 中基于关键字的条件激活
- [[**关键：**]] **指令：**模式 2（B.2 节）展示了 V4.1 早期绑定机制
- **概念引用：**模式 3（B.3 节）展示了限定领域的概念调用

- **SWITCH 架构（已弃用）：** 反模式 1（B.4 节）解释了为何在 V4 中将其替换

注： 附录 B 中的所有示例均直接提取自 `OG-PROMPTS/`目录（V1-V4.1），并包含实证验证结果。

A.5 函数

V **（验证）：** 确保格式合规性的输出验证层。R **（角色）：** 将任务映射到执行上下文的角色绑定函数。B **（绑定）：** 针对关键约束的早期绑定机制。$T(x)$**：** 从输入 x 中提取任务规范。

A.6 经实证确定的常数

- $S^* = 0.509$ （通过 V1-V4.1 优化确定）
- $\lambda^* = 4.16 \times 10^{-5}$ （成本—质量决策边界）
- $B_{critical} = 0.093$ （ [[CRITICAL:]]带来的质量提升测量值）
- $\gamma = 1.0,\ \delta = 0.1$ （根据 V1、V3、V4 的开销测量值进行校准）

A.7 完整的 O_s 计算示例

A.7.1 V3 架构分析

V3 提示词包含：

- SWITCH: `question_type` 包含 8 个分支情形（$C_1 = 8$）
- SWITCH: `domain_type` 包含 4 种分支情形（$C_2 = 4$）
- 无条件: `<linear_algebra_procedures>` 位于根级别（$|L_{过程}| \approx 100$词元）

A.7.2 计算

$$
\begin{aligned}
O_s(\text{V3}) &= \gamma \sum_{k=1}^{2} \ln(C_k) + \delta |L_{\text{proc}}| && (31)\\
&= 1.0 \times [\ln(8) + \ln(4)] + 0.1 \times 100 && (32)\\
&= 1.0 \times [2.08 + 1.39] + 10.0 && (33)\\
&= 3.47 + 10.0 && (34)\\
&= 13.47 && (35)
\end{aligned}
$$

A.7.3 V4 比较

V4 消除了 SWITCH 语句，并使过程条件化：

$$O_s(\text{V4}) = \gamma \times 0 + \delta \times 20 \tag{36}$$
$$= 0 + 2.0 \tag{37}$$
$$= 2.0 \tag{38}$$

开销削减：

$$\frac{O_s(\text{V3}) - O_s(\text{V4})}{O_s(\text{V3})} = \frac{13.47 - 2.0}{13.47} = 85.1\% \tag{39}$$

结构开销减少了 85%，这解释了为什么 V4 的质量（90.7%）高于 V3（86.4%），尽管两者的行数相近。

B 附录 B：模式库

本附录记录了第一阶段研究中已验证的 UCL 模式。我们还有更多尚未测试的模式，未来将予以发布。

B.1 模式 1：条件域激活

意图： 仅当在用户输入中检测到关键词时，才激活领域特定内容。**结构：**

```
^^CONDITION:keyword_list^^
[Domain-specific instructions and examples]
^^/CONDITION:keyword_list^^
```

已验证示例（来自 V4/V4.1）：

```
^^CONDITION: {{concept:problem_content:text_analysis}}
    CONTAINS "gram" OR "schmidt" OR "qr" OR "orthogonalization"^^
    <gram_schmidt_qr_factorization>
        [[TRANSFORM: {{concept:subscript_notation}}
            TO "{{concept:variable_name}} sub {{index}}"]]
    </gram_schmidt_qr_factorization>
^^/CONDITION:{{concept:problem_content}}^^

^^CONDITION: {{concept:problem_content}}
    CONTAINS "eigenvalue" OR "eigenvector" OR "determinant"^^
    <linear_algebra_notation>
        [[TRANSFORM: {{concept:eigenvalue_notation}}
            TO "lambda sub {{index}}"]]
    </linear_algebra_notation>
^^/CONDITION:{{concept:problem_content}}^^
```

验证结果：

- 模型：11/11（100% 成功）
- 令牌减少：相比始终激活基线减少 25-35%

- 质量：维持在 90.7%（与基线相比无统计学差异）
- 机制：已确认 $I_i \in \{0, 1\}$ 在 GPT-4、Claude、Gemini 等模型中的行为。

B.2 模式 2：关键输出指令

意图： 通过早期绑定强制执行严格的输出格式要求。**结构：**

```
[[CRITICAL:format_specification]]
```

已验证示例（来自 V4.1）：

```
[[CRITICAL: Your ONLY output is JSON.
Begin your response IMMEDIATELY with the opening
brace { character.
DO NOT output:
- Greeting or casual language
- Reasoning or explanation
- Meta-commentary
Internal calculations belong in scratchwork_answer
field INSIDE the JSON structure.]]
```

验证结果：

- 质量提升： $B_{critical} = 0.093$ (提升 9.3%)
- 合规性：格式遵循率为 100%，而未使用指令时为 90.7%
- 效果：在相同提示结构下，V4 为 90.7%， $\rightarrow$ V4.1 为 100%
- 机制：已确认在处理管道中进行早期绑定

B.3 模式3：概念引用

意图： 调用领域范围概念以确保语义精确度。**结构：**

```
{{concept:domain:specification}}
```

已验证示例：

```
{{concept:ai_identity:mathematical_tts_processor}}
{{concept:mathematical_expressions:all_notation_types}}
{{concept:tts_compatible_format:natural_spoken_language}}
{{concept:json_output:exclusive_format}}
{{concept:norm_notation:double_vertical_bars}}
{{concept:inner_product:angle_brackets}}
```

验证结果：

- 语义精确度：相比自然语言，消歧效果更佳
- 令牌效率：相比完整的概念解释，令牌数量减少 15-20%
- 可维护性：集中管理概念定义，便于更新
- 机制：已确认作用域查找适用于各种模型架构

B.4 反模式（已验证的失败）

反模式 1：SWITCH 架构（V1-V3）

问题： 由于无条件处理，所有分支都会被解析，无论其是否相关。

```
^^SWITCH: {{concept:question_type:problem_classification}}^^
    ^^CASE: {{concept:vector_calculus:mathematical_domain}}^^
        [[ENFORCE: {{concept:vector_notation}}]]
    ^^/CASE:{{concept:vector_calculus}}^^
    ^^CASE: {{concept:linear_algebra:mathematical_domain}}^^
        [[ENFORCE: {{concept:matrix_notation}}]]
    ^^/CASE:{{concept:linear_algebra}}^^
^^/SWITCH:{{concept:question_type}}^^
```

证据： $I_i \approx 1$ 适用于所有分支（V1 验证）。效率 $\eta = 1/D$。

反模式 2：过度规范（V2）

问题： Over-specification 化之外 S^* 会触发灾难性的 f 败。证据： 265 行提示词仅达到 2.3% 的质量。角色混淆，用任务描述代替执行。

反模式 3：过程复杂度（V3）

问题： </style}高 $|L_{\mathrm{proc}}|$ 无条件的 `<linear_algebra_procedures>`会增加开销。

证据： V3 有 $O_s = 13.47$221 行，质量仅为 86.4%；而 V4 有 132 行，质量达到 90.7%。

B.5 设计指南（经实证验证）

1. **Specification Level:** 1. 裕度 2. **条件粒度：** 每个条件包含 3-7 个关键词，以实现最佳区分度 3. **领域数量：** 每个提示词包含 5-10 个领域，可在保证良好覆盖的同时避免额外开销 4. **关键约束位置：** 谨慎使用 [[CRITICAL:]] （每个提示词使用 1-2 次），用于最高优先级约束 5. **概念范围：** 优先使用领域范围概念，而非全局概念，以减少歧义

B.6 局限性

这些模式已在以下方面得到验证：

- 域：数学文本转语音转换
- 模型：11 个大语言模型（共尝试 13 个；2 个因应用程序接口故障而排除）
- 样本：N=305个观测值
- 任务复杂度：中等（作业级数学）

泛化到其他领域（代码生成、创意写作、数据分析）需要根据第2至第4阶段路线图进行额外验证。

数据可用性

所有实验材料均公开提供：

- **UCL 基础工作。AI-Context-Document (ACD)**
https://github.com/antmikinka/AI-Context-Document
- **UCL 核心实现与验证脚本：**
https://github.com/antmikinka/Universal-Conditional-Logic
- **实验数据：** 全部 305 条模型响应（JSON 格式）
- **统计分析：** 包含完整方法论的 Python 脚本。
- **提示版本：** V1-V4.1 源代码。

两个代码仓库均采用 MIT 许可证，以实现最大程度的复用。随附复现说明，列出了确切的模型版本、API 参数和验证协议。

C语言扩展运算符与未来语法

状态：拟议 – 待验证

虽然核心运算符（条件、逻辑、格式化）已通过实证验证，我们仍引入一组扩展运算符，以应对复杂的提示架构。这些运算符目前仍处于理论阶段，将作为即将推出的静态分析工具的主要测试平台。

- **递归上下文注入（$\mathcal{R}$）：** 根据中间输出动态扩展上下文。
- **时间绑定（$\mathcal{T}$）：** 用于强制执行时序推理或依赖序列的逻辑的算子。
- **元认知标记（$\mathcal{M}$）：** 指导模型分析自身推理过程的指令（例如思维链形式化）。

详细规格和组合公式请参见附录 B。

D 研究路线图：迈向静态分析

我们的研究重点目前从手动验证运算符转向开发 UCL 工具链——一套用于在推理之前对提示词进行评估和优化的程序化工具。这与软件编译器的发展历程相似：从手动代码审查转向自动化静态分析。

第一阶段：UCL 检查器（当前）

我们目前正在开发一个基于 Python 的解析器，用于对 UCL 语法进行词法分析并执行静态检查：

- **语法验证：** 强制确保语法和运算符用法正确。
- **过度指定检测：** 自动标记超过理论饱和阈值（$S^* > 0.509$）的提示，以防止性能下降。

阶段 2：推理前优化（2026年第二季度）

开发一种算法优化器，自动将自然语言提示重构为UCL逻辑。其目标是在提交给模型之前最大化语义密度并最小化令牌使用量，从而有效地将人类意图“编译”为机器优化的指令。

阶段 3：社区标准化

建立 UCL征求意见稿（UCL-RFC）流程，以规范新增算子的流程，并维护跨模型兼容性。

E 验证协议

为确保未来新增算子的严谨性，我们定义了一套标准化的三阶段验证协议。该协议不仅要求验证输出，还需要深入分析模型的中间推理，以识别特定于架构的解释偏差。

1. **静态预验证：** 所有拟议算子都必须在 UCL 检查器中通过语法定义检查，以确保它们在推理前不会引入逻辑歧义或无限递归循环。

2. **实证推理测试：**

   - **样本量：** 每个算子至少进行 $N = 300$ 次试验，涵盖至少 3 个不同的模型系列（例如 Llama、GPT、Claude）。
   - **指标：** 与自然语言基线相比，必须证明令牌效率或输出质量有具有统计学显著性的提升（$p < 0.05$）。
   - **推理轨迹验证：** 验证不能仅依据最终输出。评估者必须检查模型的思维链（CoT），以确认 UCL 算子明确触发了预期的逻辑路径，而不是模型通过幻觉或死记硬背得出正确答案。

3. **跨架构比较分析：**

   - **差异映射：** 我们假设，UCL解释会因架构而异（例如，混合专家与稠密Transformer）。验证必须对不同模型的推理轨迹进行交叉分析，以分离出架构依赖。

   - **算子鲁棒性：**只有当拟议算子在不同架构上都能得到一致执行，证明其诉诸的是大语言模型语义处理的基本原理，而非特定的训练伪影时，才会将其视为“普遍验证”。