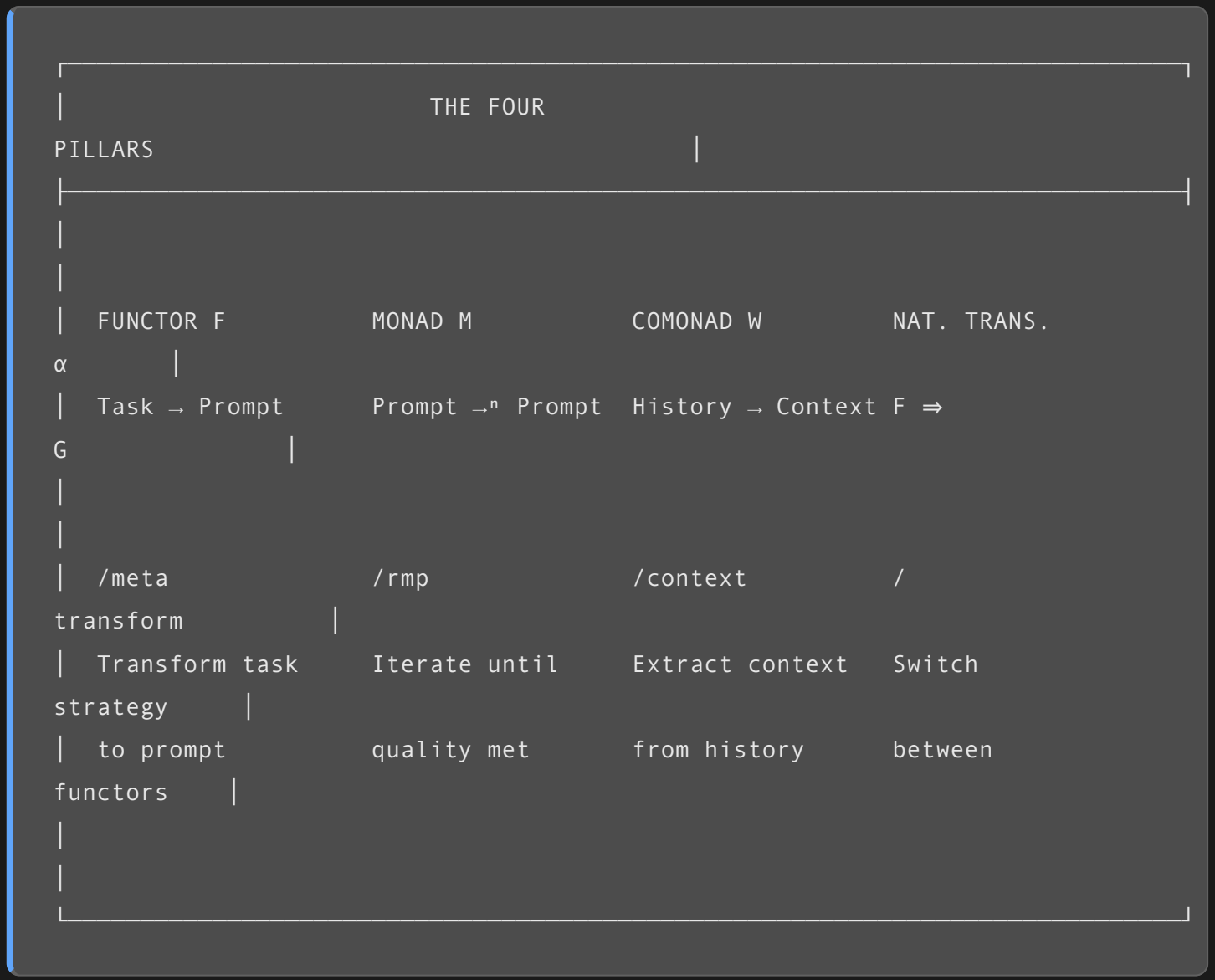
CHEAT-SHEET

1 Categorical Meta-Prompting Cheat Sheet

Version: 2.2 | **Status**: Production Ready | **Quick Reference**

1.1 Core Categorical Structures



1.2 Quick Command Reference

1.2.1 Functor F: `/meta`

```
# Basic transformation
/meta "fix the login bug"

# With domain routing
/meta @domain:SECURITY "review API endpoints"
/meta @domain:DEBUG "investigate error"
/meta @domain:ALGORITHM "optimize sort"

# With complexity tier
/meta @tier:L2 "simple task"          # Direct approach
/meta @tier:L5 "complex architecture" # Hierarchical

# With template
/meta @template:{context:expert}+{mode:cot}+{format:code} "implement
        feature"
```

1.2.2 Monad M: `/rmp`

```
# Quality-gated iteration
/rmp @quality:0.85 "implement validation"

# With max iterations
/rmp @quality:0.90 @max_iterations:5 "optimize algorithm"

# Verbose mode (show all iterations)
/rmp @mode:verbose @quality:0.80 "refine design"
```

1.2.3 Comonad W: `/context`

```
# Extract recent context
/context @mode:extract @focus:recent @depth:5 "what have we done?"

# File-focused context
/context @mode:extract @focus:file "src/auth.ts"

# Meta-observation (debug prompts)
/context @mode:duplicate "why did this fail?"

# Context-aware transformation
/context @mode:extend @transform:summarize "executive summary"
```

1.2.4 Natural Transformation α: `/transform`

```
# Strategy switch
/transform @from:zero-shot @to:chain-of-thought "explain binary search"

# Compare strategies
/transform @mode:compare @from:ZS @to:ToT "evaluate options"

# Auto-analyze best strategy
/transform @mode:analyze "debug intermittent failure"
```

1.3 Composition Operators

OPERATOR	SYMBOL	MEANING	QUALITY RULE
Sequence	→	Output A → Input B	$\min(q_1, q_2)$
Parallel	||	Run concurrently	$\text{mean}(q_1, q_2, \ldots)$
Tensor	⊗	Combine structures	$\min(q_1, q_2)$
Kleisli	>=>	Quality-gated chain	improves iteratively

1.3.1 Chain Examples

```
# Sequential (→)
/chain [/meta → /rmp @quality:0.85] "implement cache"
/chain [/context → /meta → /review] "context-aware implementation"

# Parallel (||)
/chain [/review @domain:SECURITY || /review @domain:PERFORMANCE] "audit
        code"

# Kleisli refinement (>=>)
/chain [/analyze >=> /design >=> /implement] @quality:0.85 "build feature"

# Mixed
/chain [/context → (/approach-a || /approach-b) → /synthesize] "explore
        options"
```

1.4 Modifiers Quick Reference

Modifier	Values	Default	Used With
`@mode:`	active, iterative, dry-run, spec	active	All commands
`@quality:`	0.0-1.0	0.7-0.8	/rmp, /chain
`@tier:`	L1-L7	auto	/meta
`@domain:`	ALGORITHM, SECURITY, API, DEBUG, TESTING	auto	/meta, /review
`@template:`	{context}+ {mode}+ {format}	auto	/meta
`@focus:`	recent, all, file, conversation	recent	/context
`@depth:`	1-10	3	/context
`@from:/@to:`	zero-shot, few-shot, chain-of-thought, tree-of-thought	-	/transform
`@max_iterations:`	1-10	5	/rmp

1.5 Tier Classification (L1-L7)

Tier	Tokens	Pattern	Strategy
L1	600-1200	Single operation	DIRECT
L2	1500-3000	A → B sequence	DIRECT
L3	2500-4500	design → implement → test	MULTI_APPROACH
L4	3000-6000	Parallel consensus (||)	MULTI_APPROACH
L5	5500-9000	Hierarchical with oversight	AUTONOMOUS_EVOLUTION
L6	8000-12000	Iterative loops	AUTONOMOUS_EVOLUTION
L7	12000-22K	Full ensemble	AUTONOMOUS_EVOLUTION

1.6 Quality Assessment

```
┌──────────────────────────────────────────────────────────────────────────────┐
│  DIMENSION      │  WEIGHT  │
QUESTION                                    │
├──────────────────────────────────────────────────────────────────────────────┤
│  Correctness    │   40%    │  Does it solve the
problem?                    │
│  Clarity        │   25%    │  Is it
understandable?                           │
│  Completeness   │   20%    │  Are edge cases
handled?                      │
│  Efficiency     │   15%    │  Is it well-
designed?                           │
├──────────────────────────────────────────────────────────────────────────────┤
│  Formula: q = 0.40×correctness + 0.25×clarity + 0.20×completeness
│
│             +
0.15×efficiency                                          │
└──────────────────────────────────────────────────────────────────────────────┘

Thresholds:
  ≥0.90  Excellent  │  Stop, success
  0.80-0.90  Good   │  Stop, success
  0.70-0.80  OK     │  Continue if iterative
  0.60-0.70  Poor   │  Refine
  <0.60  Failed     │  Abort or restructure
```

1.7 Common Workflows

1.7.1 Bug Fix Pipeline (W → F → F)

```
/chain [/context @mode:extract → /debug → /meta @domain:DEBUG]
        "TypeError in auth"
```

1.7.2 Code Review Pipeline (F || F || F || F)

```
/meta-review "src/auth/login.ts"
# Or manually:
/chain [/review @domain:SECURITY || /review @domain:PERFORMANCE || /review
        @domain:CORRECTNESS] "file.ts"
```

1.7.3 Feature Implementation (W → α → F → M)

```
/chain [
  /context @mode:extract @focus:all →
  /transform @mode:analyze →
  /meta @tier:L4 →
  /rmp @quality:0.88
] "implement OAuth2 authentication"
```

1.7.4 Generate Then Refine (F → M)

```
/chain [/meta @tier:L3 → /rmp @quality:0.85] "implement LRU cache"
```

1.7.5 Context-Aware Generation (W → F)

```
/chain [/context @mode:extract → /meta] "add validation following project
        patterns"
```

1.7.6 Strategy Upgrade Then Refine (α → M)

```
/chain [/transform @to:chain-of-thought → /rmp @quality:0.85] "design
        database schema"
```

1.8 Template Components

1.8.1 Context Components

```
{context:expert}    "You are an expert with deep knowledge"
{context:teacher}   "You are a patient teacher explaining step by step"
{context:reviewer}  "You are a critical reviewer looking for issues"
{context:debugger}  "You are a systematic debugger isolating problems"
```

1.8.2 Mode Components

```
{mode:direct}     "Provide a direct, concise answer"
{mode:cot}        "Think step by step before answering"
{mode:multi}      "Consider multiple approaches, then synthesize"
{mode:iterative}  "Attempt, assess, refine until quality threshold met"
```

1.8.3 Format Components

```
{format:prose}       "Write in clear paragraphs"
{format:structured}  "Use headers, lists, and tables"
{format:code}        "Provide working code with comments"
{format:checklist}   "Provide actionable checklist items"
```

1.9 Strategy Registry

Strategy	Quality Baseline	Token Cost	Best For
zero-shot	0.65	Low	Simple queries
few-shot	0.78	Medium	Pattern matching
chain-of-thought	0.85	Medium-High	Reasoning tasks
tree-of-thought	0.88	High	Search/exploration
self-consistency	0.82	High	Robustness
meta-prompting	0.90	Variable	Adaptive tasks

1.10 Categorical Laws (Must Hold)

```
FUNCTOR:
  F(id) = id                              Identity
  F(g ∘ f) = F(g) ∘ F(f)                  Composition

MONAD:
  return >=> f = f                         Left identity
  f >=> return = f                         Right identity
  (f >=> g) >=> h = f >=> (g >=> h)    Associativity

COMONAD:
  extract ∘ duplicate = id                 Left identity
  fmap extract ∘ duplicate = id            Right identity
  duplicate ∘ duplicate = fmap duplicate ∘ duplicate  Associativity

NATURAL TRANSFORMATION:
  α_B ∘ F(f) = G(f) ∘ α_A                 Naturality condition
```

1.11 Checkpoint Format

```
CHECKPOINT_[TYPE]_[N]:
  command: /[command]
  iteration: [n]
  quality:
    correctness: [0-1]
    clarity: [0-1]
    completeness: [0-1]
    efficiency: [0-1]
    aggregate: [0-1]
  quality_delta: [+/- from previous]
  budget:
    used: [tokens]
    remaining: [tokens]
  status: [CONTINUE | CONVERGED | MAX_ITERATIONS | HALT]
```

1.12 Quick Recipes

Task	Recipe
Quick fix	`/meta "fix bug"`
Quality implementation	`/rmp @quality:0.85 "implement feature"`
Context-aware work	`/chain [/context → /meta] "task"`
Strategy optimization	`/transform @to:chain-of-thought "complex problem"`
Full pipeline	`/chain [/context → /transform → /meta → /rmp] "feature"`
Code review	`/meta-review "file.ts"`
Debug issue	`/chain [/context → /debug → /meta @domain:DEBUG] "error"`
Compare approaches	`/chain [/approach-a || /approach-b] @mode:compare "problem"`

1.13 Troubleshooting

Issue	Solution
Quality not improving	Check if at fixed-point (plateau); accept or restructure
Budget exceeded	Reduce tier: `/meta @tier:L4 @budget:15000`
Unknown modifier	Valid: @mode, @quality, @tier, @budget, @domain, @template, @focus, @depth
Not converging	Lower threshold or increase @max_iterations
Wrong strategy	Use `/transform @mode:analyze` to find optimal

1.14 File Locations

```
categorical-meta-prompting/
├── CLAUDE.md                    # Framework reference
├── docs/
│   ├── CHEAT-SHEET.md          # This file
│   ├── EXAMPLES-LIBRARY.md      # 22+ validated examples
│   ├── UNIFIED-SYNTAX-SPEC.md   # Complete grammar
│   └── ARCHITECTURE-UNIFIED.md  # System design
└── .claude/
    └── commands/
        ├── meta.md              # Functor F
        ├── rmp.md               # Monad M
        ├── context.md           # Comonad W
        ├── transform.md         # Natural Trans. α
        ├── chain.md             # Composition
        └── meta-review.md       # Multi-pass review
```

Framework Version: 2.2 | **Tests**: 10/10 Passed | **Coverage**: 85%

Print this sheet for quick reference during prompt engineering sessions.