Agentic AI Patterns

Thank you to our Sponsors!

Ron Dagdag

Code > Create > Coach > Repeat.

"Opinions expressed here are my own.
I am not an agent of my employer"

R&D Engineering Manager at

7-ELEVEN

BACHELOR'S THESES

CONCAT: A programming tool for constructing collaborative agents

Mong Chin Chan
Chiu Wo Choi
Ron Lyle G. Dagdag
Gene Dexter T. Yu

Date of Publication
1999

Document Type
Bachelor's Thesis

Degree Name
Bachelor of Science in Computer Science

https://animorepository.dlsu.edu.ph/etd_bachelors/14206/

Key Takeaways

- Guidelines for robust agent design
  - **Simplicity**: Compose small, testable patterns
  - **Transparency**: Explicit planning and reasoning
  - **Responsibility**: Robust testing and monitoring

Fuzzy inputs, transformation, outputs

	Traditional Software Development	AI Software Development
INPUTS	- Text with defined set (string) - Numeric (int, float)	**Fuzzy inputs**: Open ended text - Tabular data, markdown, text, math operation
TRANSFORMATION	- Math Calculations - If, else, else if - For/while loops	**Fuzzy transformation**s: - Extract list of key words - Rewrite as paragraph - Answer a question - Brainstorm new ideas - Perform logic/math reasoning
OUTPUTS	- Text with defined set - Numeric (int, float)	**Fuzzy output**: text - Paragraph - Number(s) - JSON / Markdown
Notes	- Can be replicated	- Probabilistic: can be different every time

Traditional Software Engineering

AI Software Engineering

Strengths and Weakness of Software

Type	Key Activity	Predictability	Flexibility	Computational Complexity
Code (Software 1.0)	Write explicit instructions via computer code	Know exactly what will happen and why	Only do what you tell it	Lowest
ML (Software 2.0)	Curate high-quality training examples	Eh… kinda… sorta…	Learn from anything you teach it	Higher
Agents (Software 3.0)	Craft context to adapt model behavior	Roll the dice. Good luck lol	Can do things you didn’t realize	Highest!

3 Waves of AI

Predictive AI

Focus on analyzing data to predict outcomes.

Analyze past data to predict future outcomes.

Significance: Enabled data-driven decision-making.

Generative AI

Focus on creating content from data.

Creating content (text, images, code, videos).

Significance : Empowered creativity and productivity.

Agentic AI

Focus on autonomous actions and learning iteratively.

Autonomous actions, environment interaction, iterative learning.

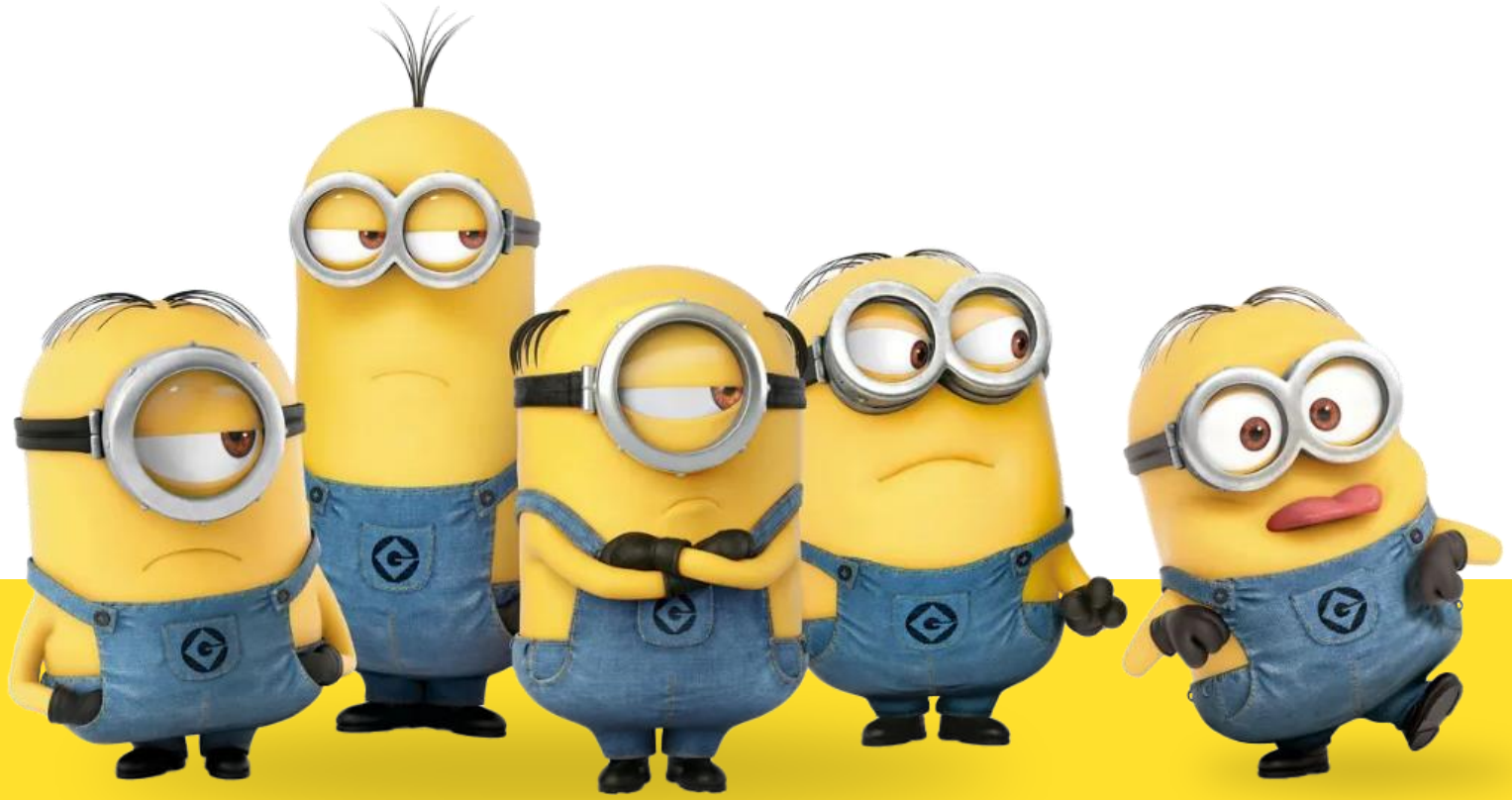
Significance : AI becomes proactive, managing complex tasks.

Agentic AI Spectrum

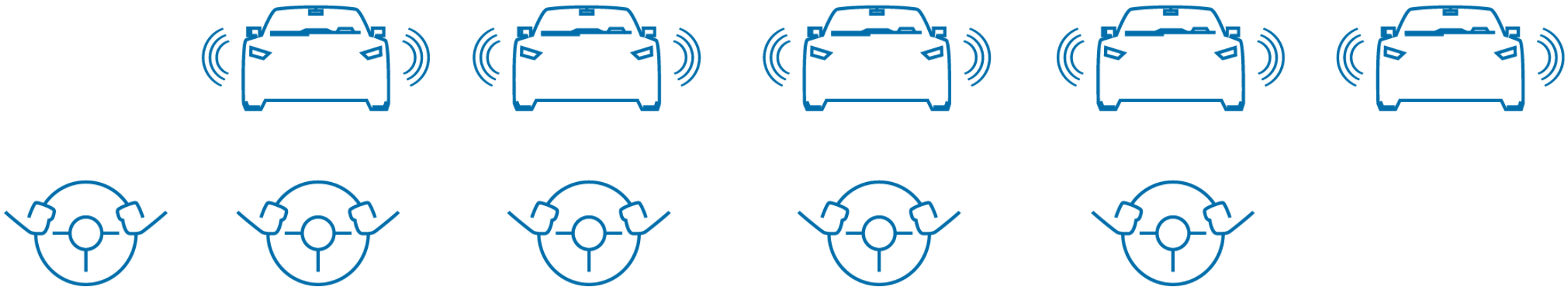
Focus on degrees of autonomy and usefulness.

Defined Path / Workflow

Fully Autonomous



FIVE LEVELS OF VEHICLE AUTONOMY



LEVEL 0

No Automation: the driver is in complete control of the vehicle at all times.

LEVEL 1

Driver Assistance: the vehicle can assist the driver or take control of either the vehicle's speed, through cruise control, or its lane position, through lane guidance.

LEVEL 2

Occasional Self-Driving: the vehicle can take control of both the vehicle's speed and lane position in some situations, for example on limited-access freeways.

LEVEL 3

Limited Self-Driving: the vehicle is in full control in some situations, monitors the road and traffic, and will inform the driver when he or she must take control.

LEVEL 4

Full Self-Driving Under Certain Conditions: the vehicle is in full control for the entire trip in these conditions, such as urban ride-sharing.

LEVEL 5

Full Self-Driving Under All Conditions: the vehicle can operate without a human driver or occupants.

Source: SAE & NHTSA

Levels of autonomy in LLM applications


code


LLMs

			Decide Output of Step	Decide Which Steps to Take	Decide What Steps are Available to Take
HUMAN-DRIVEN	1	Code			
	2	LLM Call	*one step only*		
	3	Chain	*multiple steps*		
	4	Router		*no cycles*	
AGENT-EXECUTED	5	State Machine		*cycles*	
	6	Autonomous			

LangChain

What is an AI Agent?

Semi-autonomous software that uses tools to achieve goals.

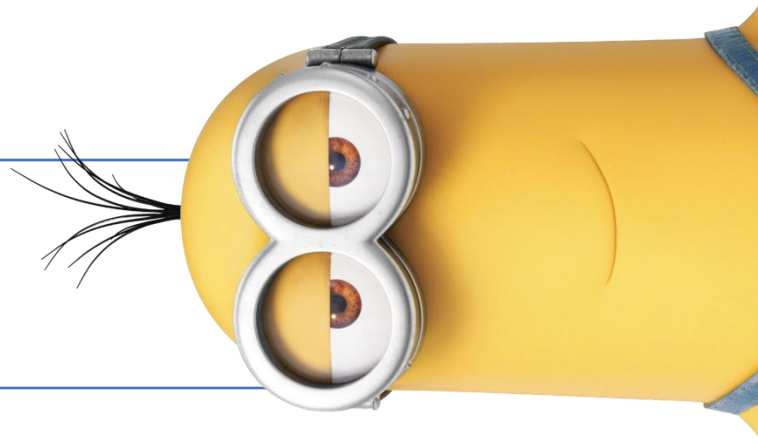
Essential Skills for Building AI Agents

Data plumbing & Integrations (Tool use and MCP)

Prompt engineering

Evaluation and Observability

Coding—still critical!



Why Learn Agentic Patterns?

Shared vocabulary

→ faster design reviews and debugging.

Composable building blocks

→ mix & match for real apps.

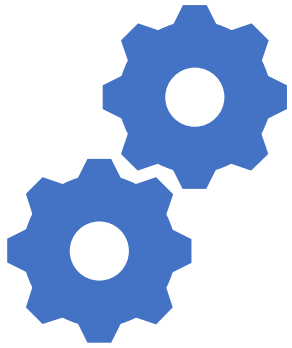
Safer, more reliable agents

→ guardrails and evaluation loops.

Cloud-native scale

→ state, queues, workflows, and protocols.

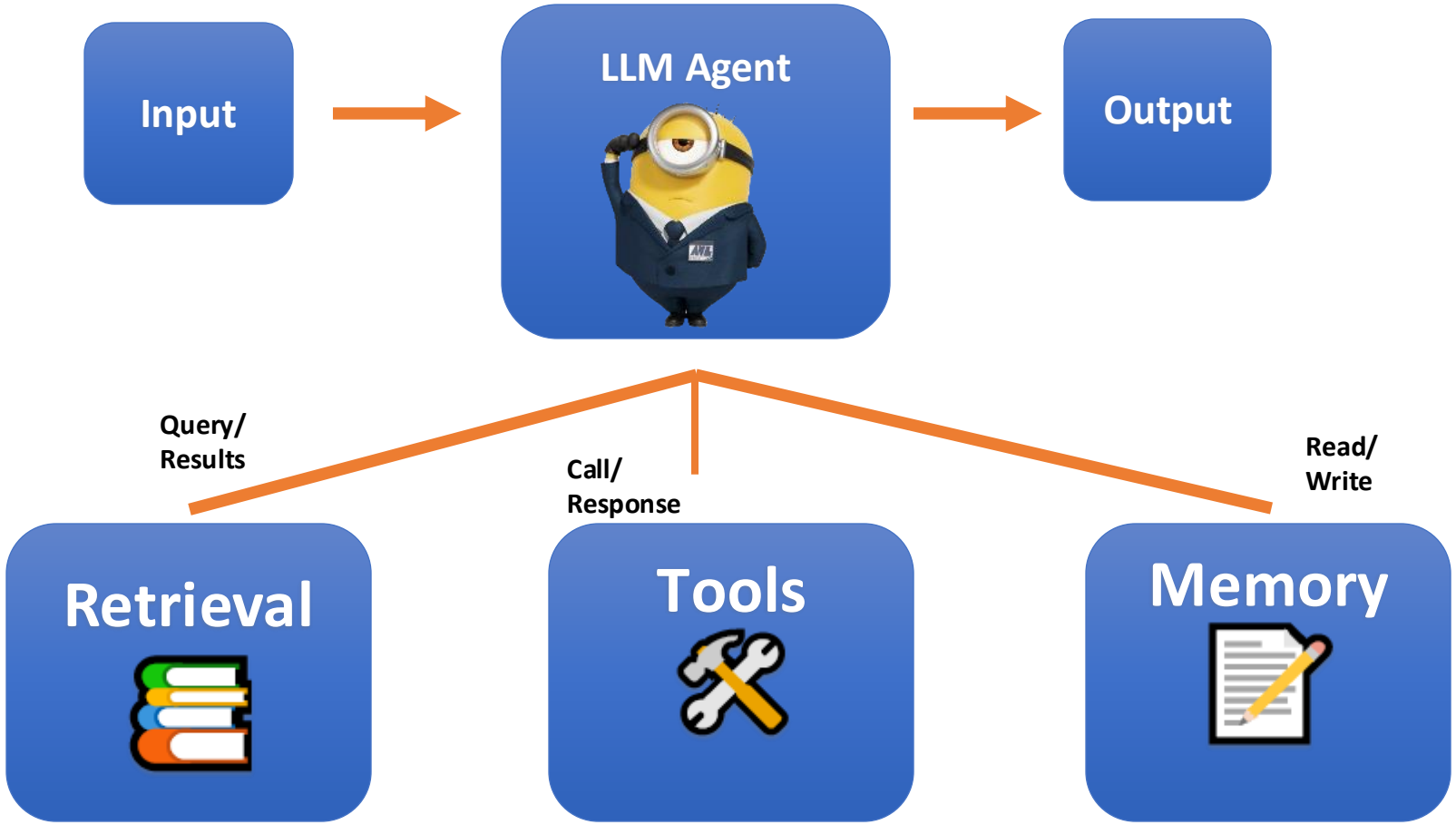
Assemble the Minions



Combine specialized tools quickly instead of reinventing everything

Modular design accelerates innovation

Augmented LLM



Context Layer

Augmented LLM Demo

Azure AI Foundry Agents

AI Agent Workflows

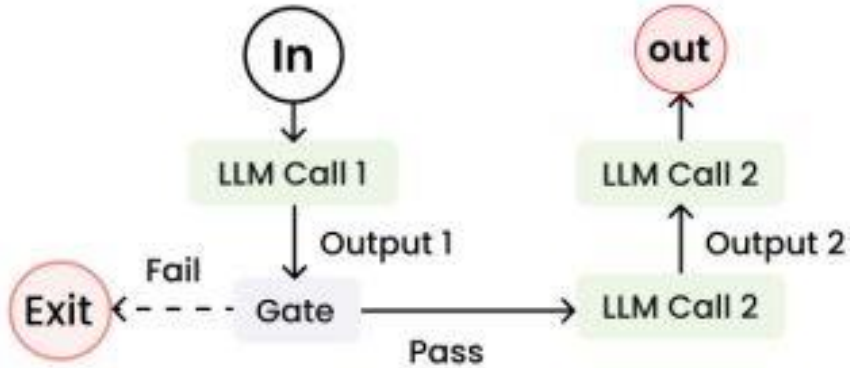
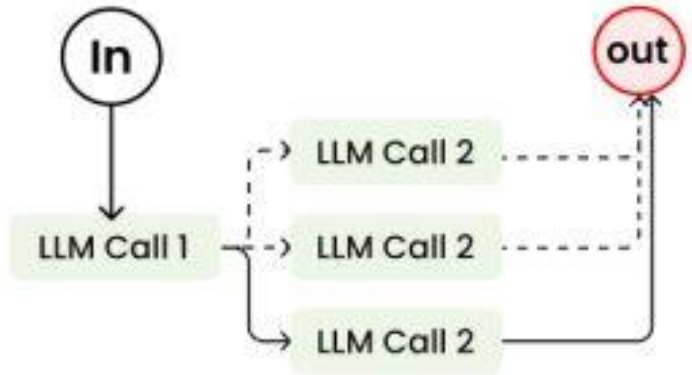
- Linear Workflow / Human-Driven Checklist
  - Chaining
  - Routing
  - Parallelization (Sectioning, Voting)
- LLM-Driven / Agent-Executed
  - Orchestrator-Workers
  - Evaluator-Optimizer (Self-reflection loops)

https://www.anthropic.com/engineering/building-effective-agents

ANTHROP\C

@rakeshgohel01

LLM based AI Agent Workflows with code </>

Workflow Name	Diagram	Use cases
Prompt Chaining		For Copywriting, Proof reading and very niche tasks
Routing		For Agent Retrieval tasks like CRM Agents Agentic RAG and much more

Chaining

- Sequential steps with clear state hand-off
- Great for multi-part questions or stepwise tasks
- Keeps prompt context focused and compact
- Use Case: For Copywriting, proof-reading and very niche tasks

Chaining

Chaining Demo

Chaining Demo

```python
async def chain(input: str, prompts: List[str]) -> str:
    """Chain multiple LLM calls sequentially, passing results between steps."""
    agent = Agent(AI_MODEL)
    result = input
    for i, prompt in enumerate(prompts, 1):
        agent = Agent(AI_MODEL, system_prompt=prompt)
        response = await agent.run(f"nInput:\n{result}")
        result = response.output
        show(result, title=f"Step {i}")
    return result
```

```
data_processing_steps = [
    """Extract only the numerical values and their associated minion metrics from the text.
    Format each as 'value: minion metric' on a new line.
    Example format:
    92: banana consumption rate
    45%: "BELLO!" frequency increase""",

    """Convert all numerical values to percentages where possible for minion analytics.
    If not a percentage or points, convert to decimal (e.g., 92 bananas -> 92%).
    Keep one number per line.
    Example format:
    92%: banana consumption rate
    45%: "BELLO!" frequency increase""",

    """Sort all lines in descending order by numerical value for minion performance ranking.
    Keep the format 'value: minion metric' on each line.
    Example:
    92%: banana consumption rate
    87%: yellow outfit compliance""",

    """Format the sorted minion data as a markdown table with columns:
    | Minion Metric | Value |
    |:--|--:|
    | Banana Consumption Rate | 92% |"""
]

report = """
Q3 Minion Performance Summary:
Our minion happiness score rose to 92 points this quarter after the banana supply increase.
"BELLO!" frequency grew by 45% compared to last year's metrics.
Yellow outfit compliance is now at 23% improvement in our primary lab.
Minion distraction incidents decreased to 5% from 8% after removing shiny objects.
New evil scheme participation cost is $43 per minion recruit.
Gadget adoption rate increased to 78% among tech-savvy minions.
Gru satisfaction with minions is at 87 points.
Banana storage efficiency improved to 34% reduction in spoilage.
""".strip()

show(report, title="Input text")
formatted_result = await chain(report, data_processing_steps)
show(formatted_result, title="Result")
```

Routing

- Use heuristics, small classifiers, or LLM scoring.
- Branch logic chooses the right tool or sub-agent
- Ideal for multi-modal input or domain dispatch
- Log routing decisions for observability.
- Use Case: For Agent Retrieval Tasks like Agentic RAG, CRM Agents

Routing

Routing Demo

```python
class RouteSelection(BaseModel):
    reasoning: str = Field(..., description=(
        'Brief explanation of why this ticket should be routed '
        'to a specific team. Consider key terms, user intent, '
        'and urgency level.'
    ))
    selection: str = Field(..., description='The chosen team name')


async def route(input: str, routes: Dict[str, str]) -> str:
    """Route input to specialized prompt using content classification."""

    # First, determine appropriate route using LLM with chain-of-thought
    show(f"{list(routes.keys())}", title="Available Routes")

    routing_agent = Agent(
        AI_MODEL,
        system_prompt=(
            'Analyze the input and select the most appropriate support team '
            f'from these options: {list(routes.keys())}'
        ),
        output_type=RouteSelection,
    )
    route_response = await routing_agent.run(input)
    reasoning = route_response.output.reasoning
    route_key = route_response.output.selection.strip().lower()

    show(reasoning, title="Routing Analysis")
    show(f"{route_key}", title="Selected Route")

    # Process input with selected specialized prompt
    worker_agent = Agent(AI_MODEL, system_prompt=routes[route_key])
    return (await worker_agent.run(input)).output
```

```
support_routes = {
    "banana": """You are a banana supply specialist at Gru's lab. Follow these guidelines:
    1. Always start with "Banana Supply Response:"
    2. First acknowledge the specific banana-related issue
    3. Explain banana inventory, quality, or delivery problems clearly
    4. List concrete next steps with timeline for banana resolution
    5. End with alternative banana options if relevant

    Keep responses enthusiastic but professional about banana matters.""",

    "gadget": """You are a minion gadget technical support engineer. Follow these guidelines:
    1. Always start with "Gadget Support Response:"
    2. List exact steps to resolve the gadget malfunction
    3. Include safety requirements for minion operation
    4. Provide workarounds for common gadget problems
    5. End with escalation path to Dr. Nefario if needed

    Use clear, numbered steps and mention "BELLO!" for encouragement.""",

    "security": """You are a minion security specialist for Gru's operations. Follow these guidelines:
    1. Always start with "Security Support Response:"
    2. Prioritize lab security and minion identification verification
    3. Provide clear steps for access recovery/changes to restricted areas
    4. Include security tips and warnings about minion imposters
    5. Set clear expectations for resolution time

    Maintain a serious, security-focused tone while being minion-friendly.""",

    "training": """You are a minion training specialist. Follow these guidelines:
    1. Always start with "Training Support Response:"
    2. Focus on minion skill development and best practices
    3. Include specific examples of proper minion behavior
    4. Link to relevant training manual sections
    5. Suggest related skills that might help with evil schemes

    Be educational and encouraging, use minion language occasionally."""
}
```

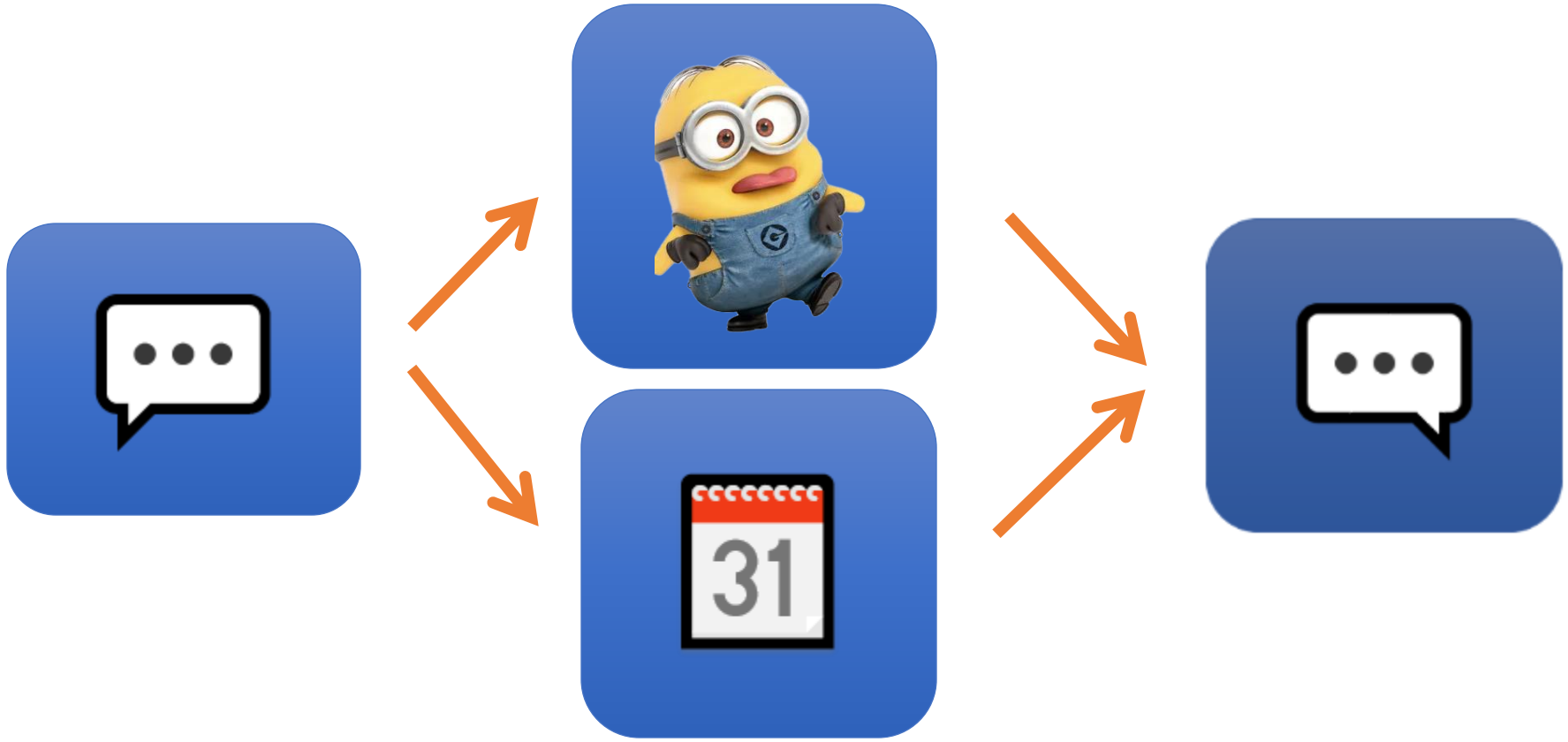
Routing Demo

```
# Test with different minion support tickets
tickets = [
    """Subject: Can't access the secret lab
    Message: BELLO! I've been trying to get into the secret lab for the past hour but the scanner
    keeps saying 'unrecognized minion' error. I'm sure I'm the right minion! Can you help me regain
    access? This is urgent as I need to prepare the freeze ray by end of day.
    - Minion Kevin""",

    """Subject: Banana shortage in cafeteria
    Message: Hello, I just noticed we're completely out of bananas in the minion cafeteria, but I
    thought we had a fresh shipment scheduled for today. The other minions are getting restless
    without their banana breaks. Can you explain this shortage and fix it?
    Thanks,
    Minion Stuart""",

    """Subject: How to operate the new shrink ray?
    Message: I need to learn how to operate the new shrink ray gadget for tomorrow's mission. I've
    looked through the manual but can't figure out the safety protocols. Is there a training
    session available? Could you walk me through the steps?
    Best regards,
    Minion Bob"""
]

print("Processing minion support tickets...\n")
for i, ticket in enumerate(tickets, 1):
    show(ticket, title=f"Ticket {i}")
    response = await route(ticket, support_routes)
    show(response, title=f"Response {i}")
```

Parallelization

- Run independent subtasks concurrently for speed.
- Two flavors: vote (merge results) or section(split workload)
- Aggregate results with validation or voting.
- Mind rate limits and shared-state conflicts.
- Use Case: Multi-Agent tasks like Coding tasks, Vulnerabilities detection and much more

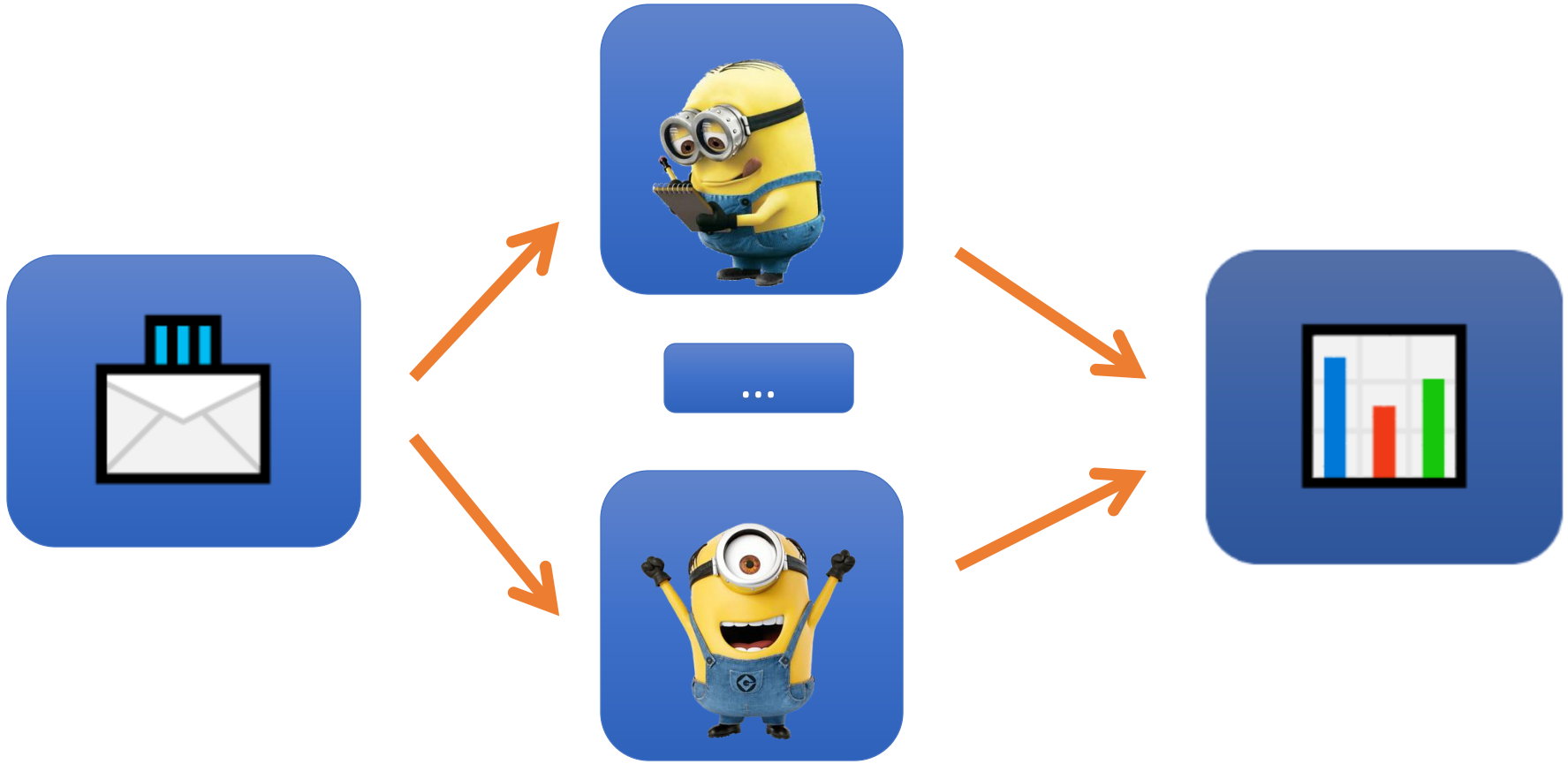
Parallelization



Parallelization

(Sectioning)

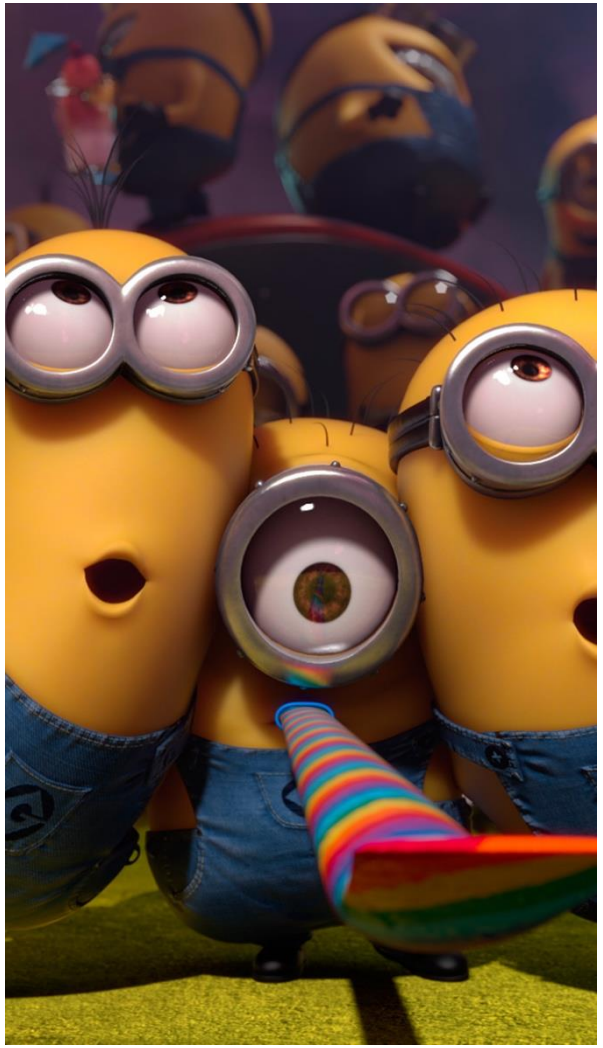
Parallelization



Parallelization

(Voting)

Parallelization Demo



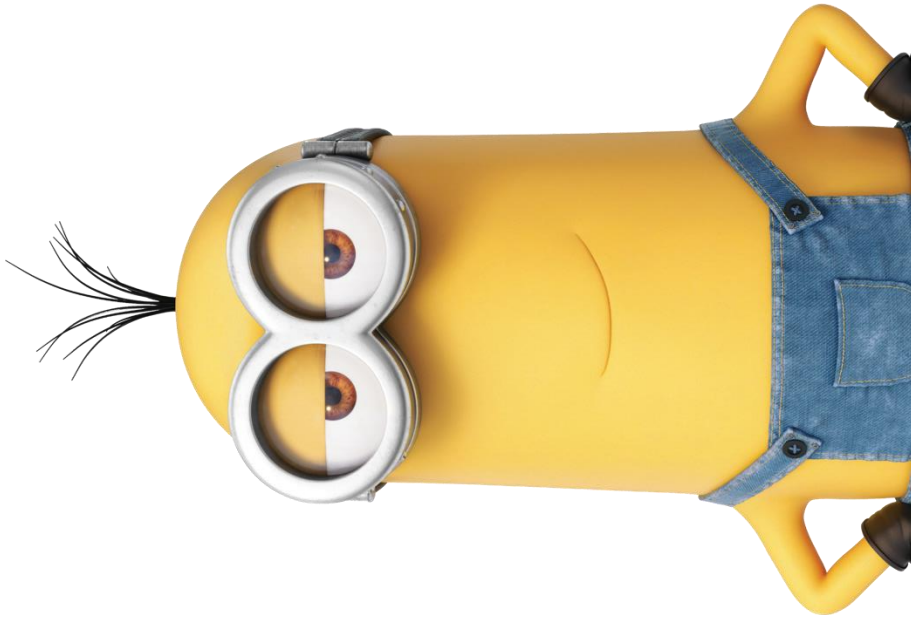
Parallelization Demo

```python
async def parallel(prompt: str, inputs: List[str]) -> List[str]:
    """Process multiple inputs concurrently with the same prompt."""
    agent = Agent(AI_MODEL, system_prompt=prompt)
    results = await asyncio.gather(*[
        agent.run(input)
        for input in inputs
    ])
    return [result.output for result in results]
```

```
stakeholders = [
    """Minions:
    - Banana supply dependent
    - Want latest gadgets
    - Safety concerns with explosions""",

    """Lab Scientists (Dr. Nefario):
    - Equipment reliability worries
    - Need advanced technology
    - Want clear evil scheme direction""",

    """Gru (Boss):
    - Expects successful missions
    - Want cost-effective operations
    - Reputation risks with failed schemes""",

    """Gadget Suppliers:
    - Production capacity constraints
    - Price pressures from competition
    - Technology upgrade transitions"""
]

impact_results = await parallel(
    """Analyze how changes in the evil villain industry will impact this stakeholder group.
    Provide specific impacts and recommended actions for Gru's operations.
    Format with clear sections and priorities.""",
    stakeholders
)

for stakeholder, result in zip(stakeholders, impact_results):
    show(result, stakeholder.split(':')[0])
```

Dynamic & Autonomous Agents

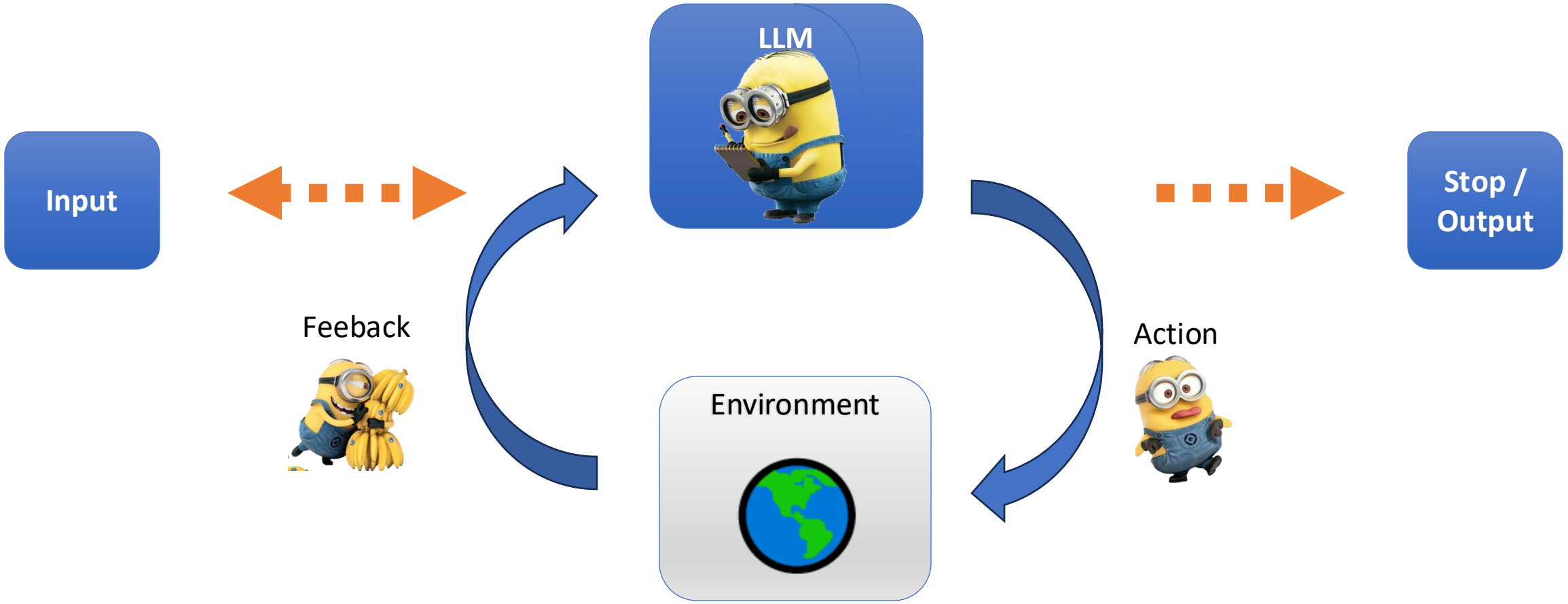
- Agents plan and act independently, using environment feedback.
  - Understand complex inputs
  - Reason and plan steps
  - Use tools reliably
  - Recover from errors



Agent Loop

1. Receive task or instruction
2. Plan the steps
3. Execute a step (often with tools)
4. Observe results (ground truth)
5. Assess progress
6. Repeat until done or stop condition

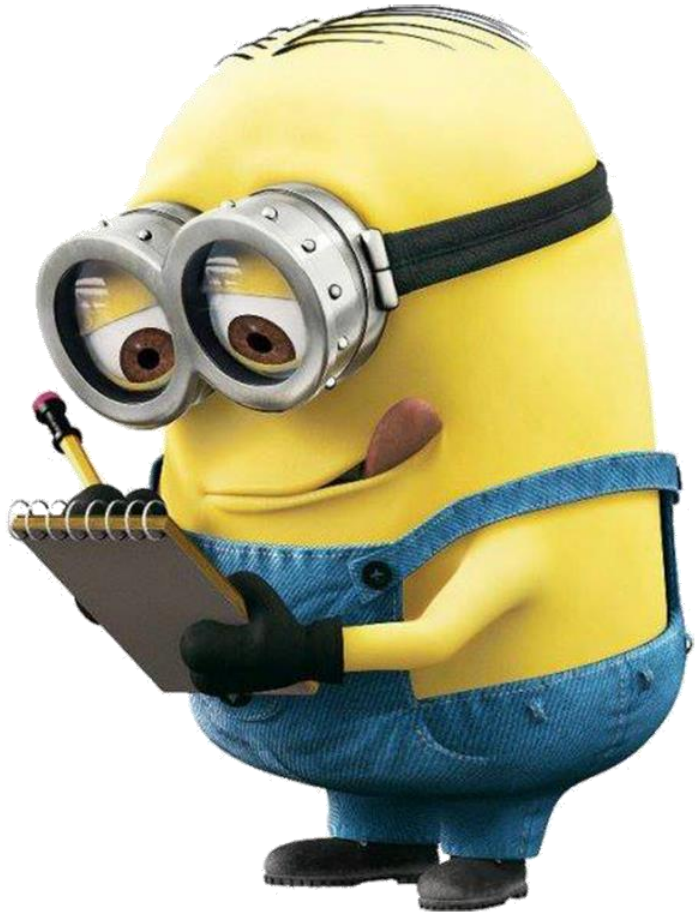
Agent Loop



Key Building Blocks

- Memory → maintains context across interactions
- Tool usage → APIs, DBs, web search
- Dynamic decision-making → chooses next steps
- Iterative refinement → improve through cycles

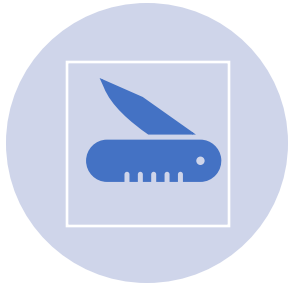
Key Elements of AI Agent



- Role Playing
- Focus
- Tools
- Cooperation / Collaboration
- Guardrails / Limitations
- Memory

When to Use AI agents?

Complex judgment, exceptions, ambiguous tasks



Rules engines that are brittle/expensive to maintain

Heavy unstructured or semi-structured data

Open-ended: difficult or impossible to predict required number of steps

Specialist Over Generalist

- Focused roles boost precision & relevance.
- Benefits: clarity, consistency, domain judgment.

Specialist Over Generalist

- When to Use:
  - Open-ended tasks with unpredictable steps
- Caveats:
  - Higher costs
  - Risk of compounding errors
  - Need sandbox testing
  - Require robust guardrails

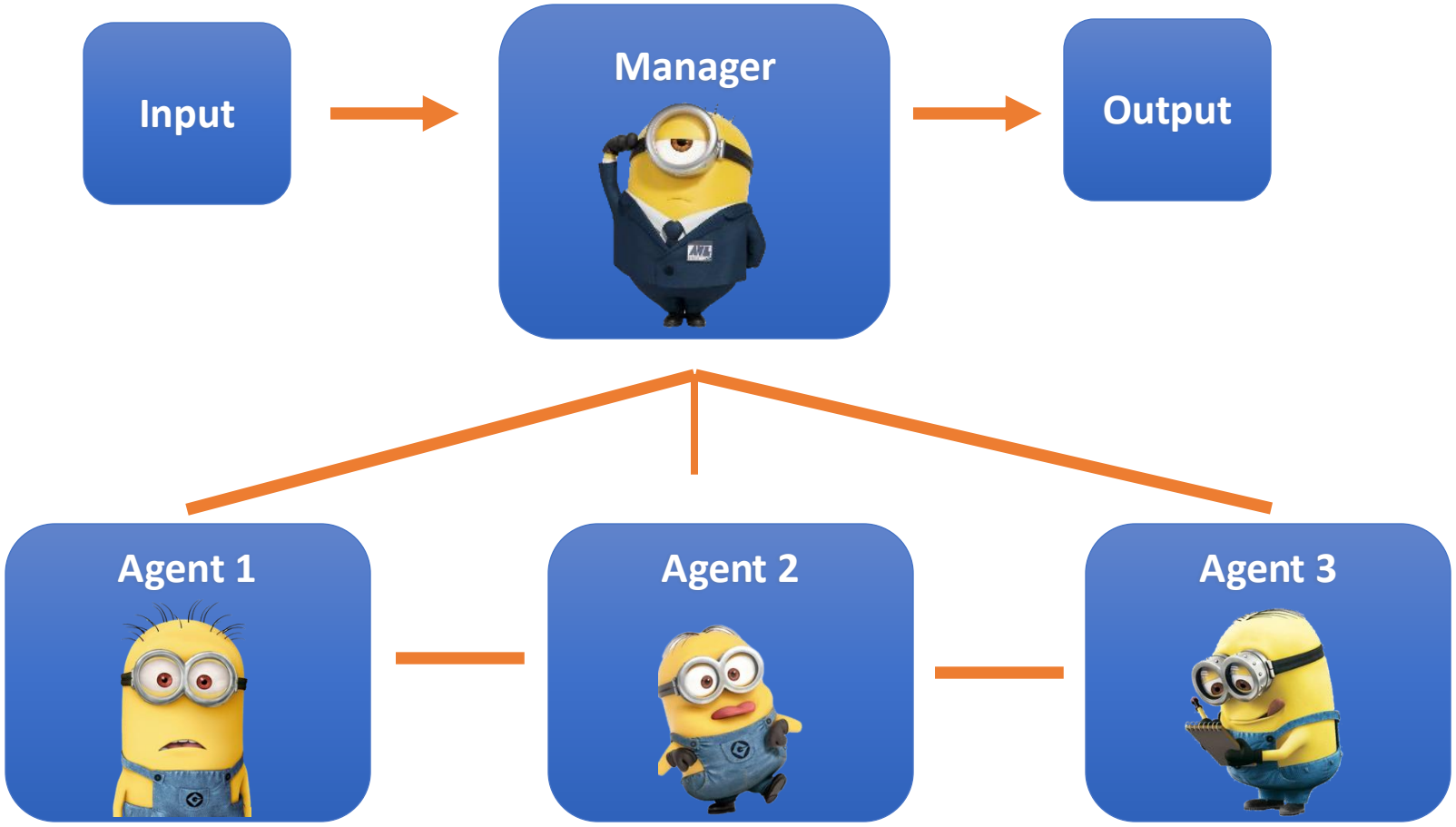
Orchestrator-Workers

- Coordinator assigns tasks; workers specialize and report back.
- Great for multi-skill teams with pluggable workers.
- Supports long-running or distributed jobs
- Add timeouts, retries, and escalation to humans.
- Use Case: For Critical Tasks like Predictive Analysis and Market Prediction and much more

Multi-Agent Collaboration

- Role-based agents (Leader, Researcher, Coder, Tester)
- Shared memory, meeting logs, and decision records
- Conflict resolution rules to avoid loops
- Agent Workflow orchestrates hand-offs & shared state
- Root agent kicks off, others update state atoms
- Great for research-write-review loops

LLM-Driven Flows



Orchestrator-Workers

Orchestrator-Workers Demo

Orchestrator-Workers Demo

```python
class Task(BaseModel):
    type: str = Field(..., description=(
        'The type of minion task approach. '
        'For example: "enthusiastic", "technical", "banana-focused", "gadget-oriented", ...'))
    description: str = Field(
        ...,
        description='Clear description for executing this minion task.'
    )


class OrchestratorResponse(BaseModel):
    analysis: str = Field(..., description=(
        'Explain your understanding of the minion task and which variations '
        'would be valuable for Gru\'s operations. Focus on how each approach serves '
        'different aspects of minion management or evil schemes.'
    ))
    tasks: List[Task] = Field(..., description="List of minion tasks")
```

Orchestrator-Workers Demo

```python
async def orchestrate(task: str) -> Dict:
    """Process minion task by breaking it down and running subtasks in parallel."""

    orchestrator_agent = Agent(
        AI_MODEL,
        system_prompt=(
            'You are Gru\'s master orchestrator for minion operations. ',
            'Analyze this minion-related task and break it down into ',
            '2-3 distinct approaches that different types of minions could handle.',
        ),
        output_type=OrchestratorResponse,
    )
    orchestrator_response = await orchestrator_agent.run(task)

    analysis = orchestrator_response.output.analysis
    tasks = orchestrator_response.output.tasks

    show('', title='Minion Orchestrator Output')
    show(analysis, title='Analysis')
    show([task.model_dump() for task in tasks], title='Minion Tasks')

    # Process all the minion tasks in parallel and collect results
    worker_agent = Agent(
        AI_MODEL,
        system_prompt='You are a specialized minion worker. Generate content based on the minion task specification. Use minion language and enthusiasm where appropriate.',
    )
    worker_responses = await asyncio.gather(*[
        worker_agent.run(json.dumps(
            {'original_minion_task': task} | task_info.model_dump()
        ))
        for task_info in tasks
    ])

    for task, response in zip(tasks, worker_responses):
        show(response.output, title=f"Minion Worker Result ({task.type})")
```

```python
await orchestrate(
    'Create a comprehensive training manual for new minions joining Gru\'s evil operations, covering banana breaks, gadget safety, and proper "BELLO!" etiquette.'
)
```

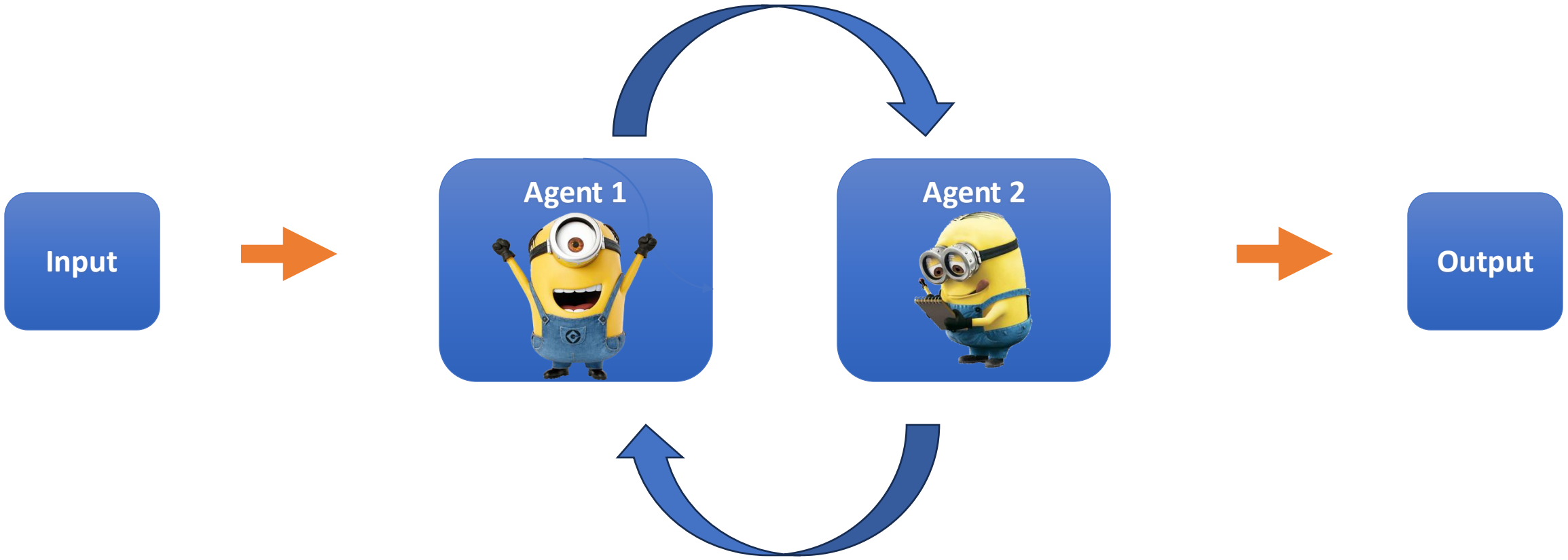
Evaluator-Optimizer (Critique Loop)

- Separate evaluator/critic scores outputs with rubrics.
- Optimizer (or main agent) revises until target score.
- Yields safer outputs and measurable quality.
- Boosts accuracy but increases cost – cache wisely
- Use Case: For Personalized Chatbots and complex search and much more

Reflection

- Agent critiques its own output and retries.
- Improves quality and reduces obvious errors.
- Use structured rubrics and stop conditions.

LLM-Driven Flows



Evaluator-Optimizer

Evaluator-Optimizer Demo

Evaluator-Optimizer Demo

```python
class GeneratorResponse(BaseModel):
    thoughts: str = Field(..., description=(
        'Your understanding of the task and feedback '
        'and how you plan to improve.'
    ))
    response: str = Field(..., description='The generated solution.')


async def generate(prompt: str, task: str, context: str = "") -> tuple[str, str]:
    """Generate and improve a solution based on feedback."""
    system_prompt = prompt
    if context:
        system_prompt += f"\n\n{context}"

    generator_agent = Agent(
        AI_MODEL,
        system_prompt=system_prompt,
        output_type=GeneratorResponse,
    )
    response = await generator_agent.run(f'Task:\n{task}')

    thoughts = response.output.thoughts
    result = response.output.response

    show('', title='Generation')
    show(thoughts, title='Thoughts')
    show(result, title='Generated')

    return thoughts, result
```

Evaluator-Optimizer Demo

```python
class EvaluatorResponse(BaseModel):
    thoughts: str = Field(..., description=(
        'Your careful and detailed review and evaluation of the submited content.'
    ))
    evaluation: str = Field(..., description='PASS, NEEDS_IMPROVEMENT, or FAIL')
    feedback: str = Field(..., description='What needs improvement and why.')


async def evaluate(prompt: str, content: str, task: str) -> tuple[str, str]:
    """Evaluate if a solution meets requirements."""
    evaluator_agent = Agent(
        AI_MODEL,
        system_prompt=f'{prompt}\n\nTask:\n{task}',
        output_type=EvaluatorResponse,
    )
    response = await evaluator_agent.run(content)
    evaluation = response.output.evaluation
    feedback = response.output.feedback

    show('', title='Evaluation')
    show(evaluation, title='Status')
    show(feedback, title='Feedback')

    return evaluation, feedback
```

Evaluator-Optimizer Demo

```python
async def loop(
    task: str, evaluator_prompt: str, generator_prompt: str
) -> tuple[str, list[dict]]:
"""Keep generating and evaluating until requirements are met."""
memory = []
chain_of_thought = []

thoughts, result = await generate(generator_prompt, task)
memory.append(result)
chain_of_thought.append({"thoughts": thoughts, "result": result})

while True:
    evaluation, feedback = await evaluate(evaluator_prompt, result, task)
    if evaluation == "PASS":
        return result, chain_of_thought

    context = "\n".join([
        "Previous attempts:",
        *[f"- {m}" for m in memory],
        f"\nFeedback: {feedback}"
    ])

    thoughts, result = await generate(generator_prompt, task, context)
    memory.append(result)
    chain_of_thought.append({"thoughts": thoughts, "result": result})
```

Combining Patterns

Combining Patterns

- **Real systems chain patterns**: Planning + Tool Use + Reflection.
- **Director–Workers** often embed Evaluator loops.
- **Routing** can decide when to use Multi-Agent vs single-agent.

When to Use Agents?

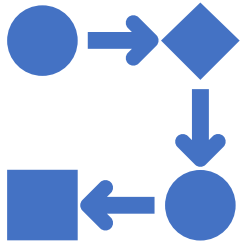
Start simple: single LLM + retrieval often suffices

Workflows: predictable, fixed code paths, low-latency, cheaper

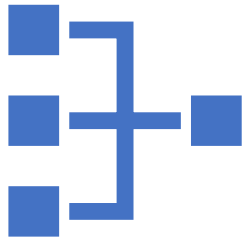
Agents: flexible, dynamic control, open-ended tasks, rapidly changing environments, decision-making at scale

Trade-off: ↑ cost & latency for ↑ task performance

Do Start
but Start Simple



Linear Workflows
(Human-Driven)



Autonomous Agents
(Agent-Executed)

Best Practices & Pitfalls

- ✅ Use LLM-driven agents when the workflow CANNOT be hard-coded.
- ⛔ Avoid them if a deterministic path solves the task reliably.
- Monitor cost, latency, and hallucination rates
- Version prompts and track metrics
- Use reflection loops sparingly
- Start simple; layer complexity gradually

Common Mistakes to Avoid

Common Mistakes to Avoid

- Unclear instructions.
- “God” tasks.
- Misaligned description/output.
- Process not understood.
- Premature hierarchies.
- Vague agents.

Guardrails & Safety

- Policies, allow/deny lists, data-loss prevention.
- Tool whitelists with caps; human-in-the-loop gates.
- Structured audits: traces, decisions, and tool calls.

Key Takeaways

- Guidelines for robust agent design
  - **Simplicity**: Compose small, testable patterns, start with linear workflows
  - **Transparency**: Explicit planning and reasoning, specialist over generalists
  - **Responsibility**: Robust testing and monitoring, guardrails and safety, human-in-the-loop gates

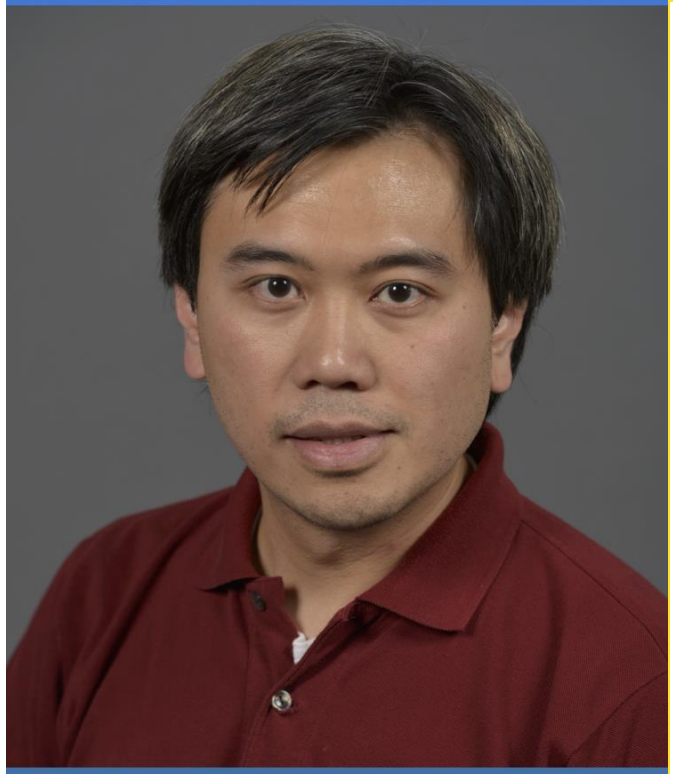
banana

https://github.com/rondagdag/agentic-ai-patterns-talk

- The Minions' favorite word and a symbol of their playful nature.

Ron Dagdag

Microsoft® Most Valuable Professional



Award Categories
AI, Windows Development,
Internet of Things, Mixed Reality

- **R&D Engineering Manager at 7-Eleven**
- **10th year Microsoft MVP awardee**
- **Linked In**
  www.linkedin.com/in/rondagdag/
- **www.dagdag.net**
- **@rondagdag**
- **Thanks for geeking out on Agentic AI Patterns**

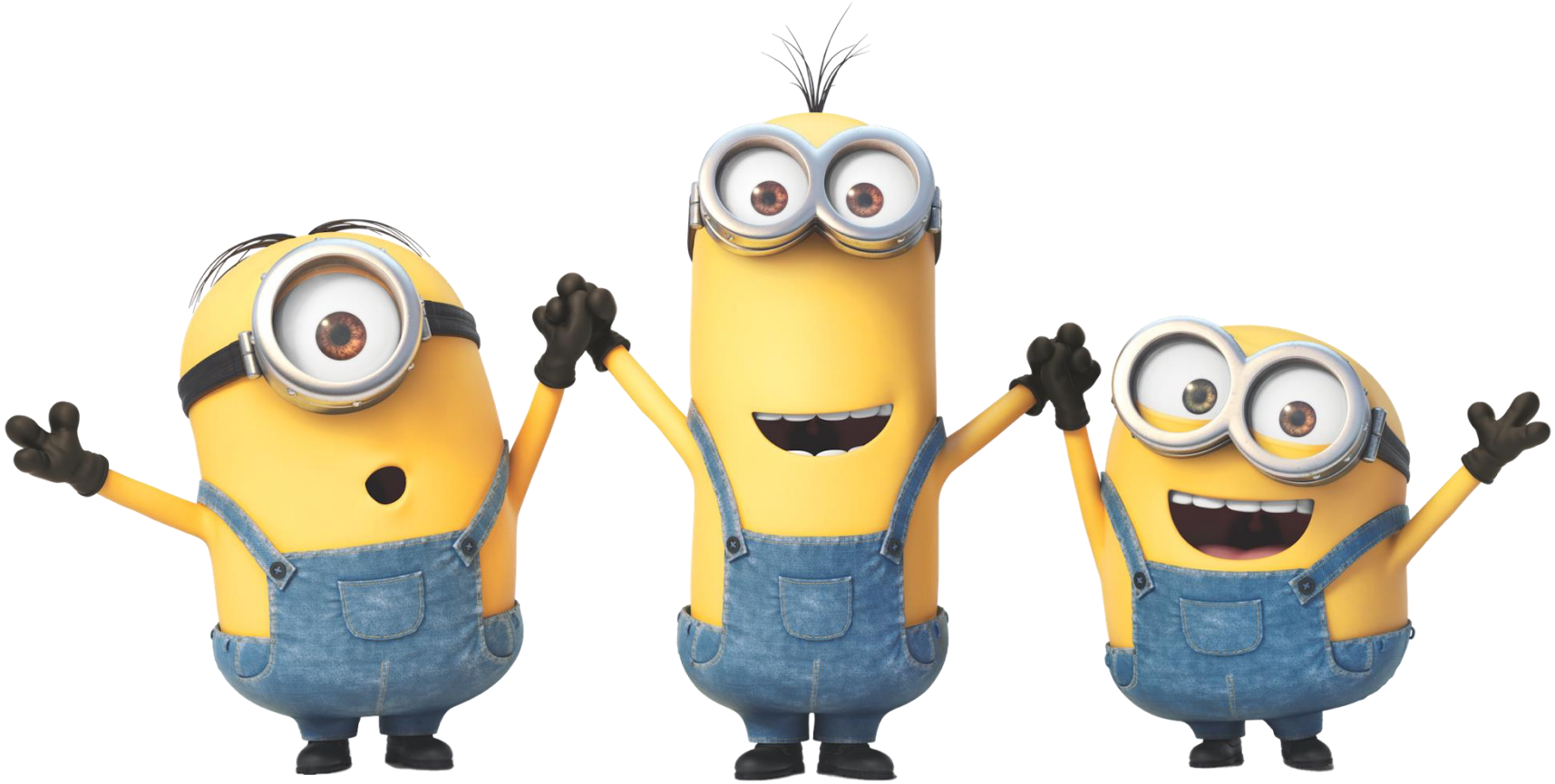
Illumination Entertainment

Illumination Entertainment is an American computer animation studio founded by Chris Meledandri in 2007. The studio is co-owned by Universal Pictures, a division of Comcast through its wholly owned subsidiary NBCUniversal. Illumination has produced several successful animated films, including the Despicable Me franchise and minions

thank-you

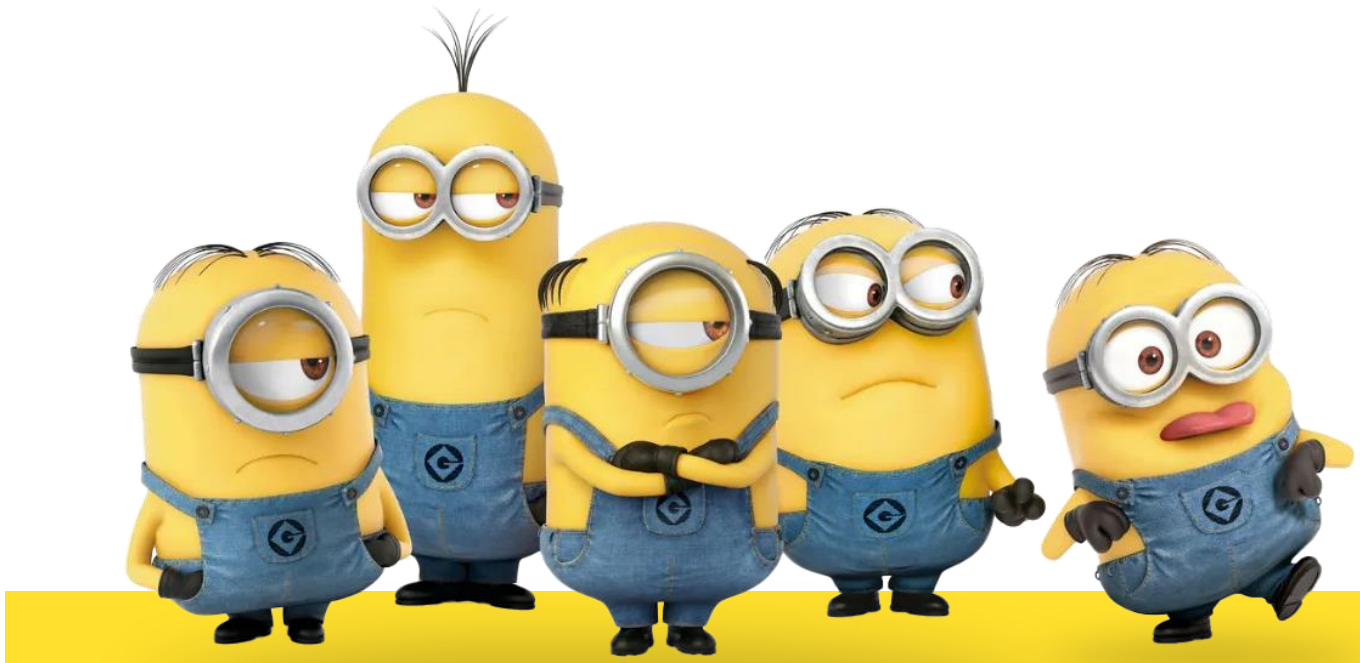
Introduction to "Minions"

The "Minions" movie, produced by Illumination Entertainment, is a spin-off of the "Despicable Me" franchise. Released in 2015, it focuses on the endearing Minions, small, yellow creatures with a penchant for misadventures.



Agents and Automation

The movie features various settings, from the bustling streets of New York City to the grandeur of Villain-Con. The richly detailed locations enhance the storytelling.



Meet the team

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Stuart

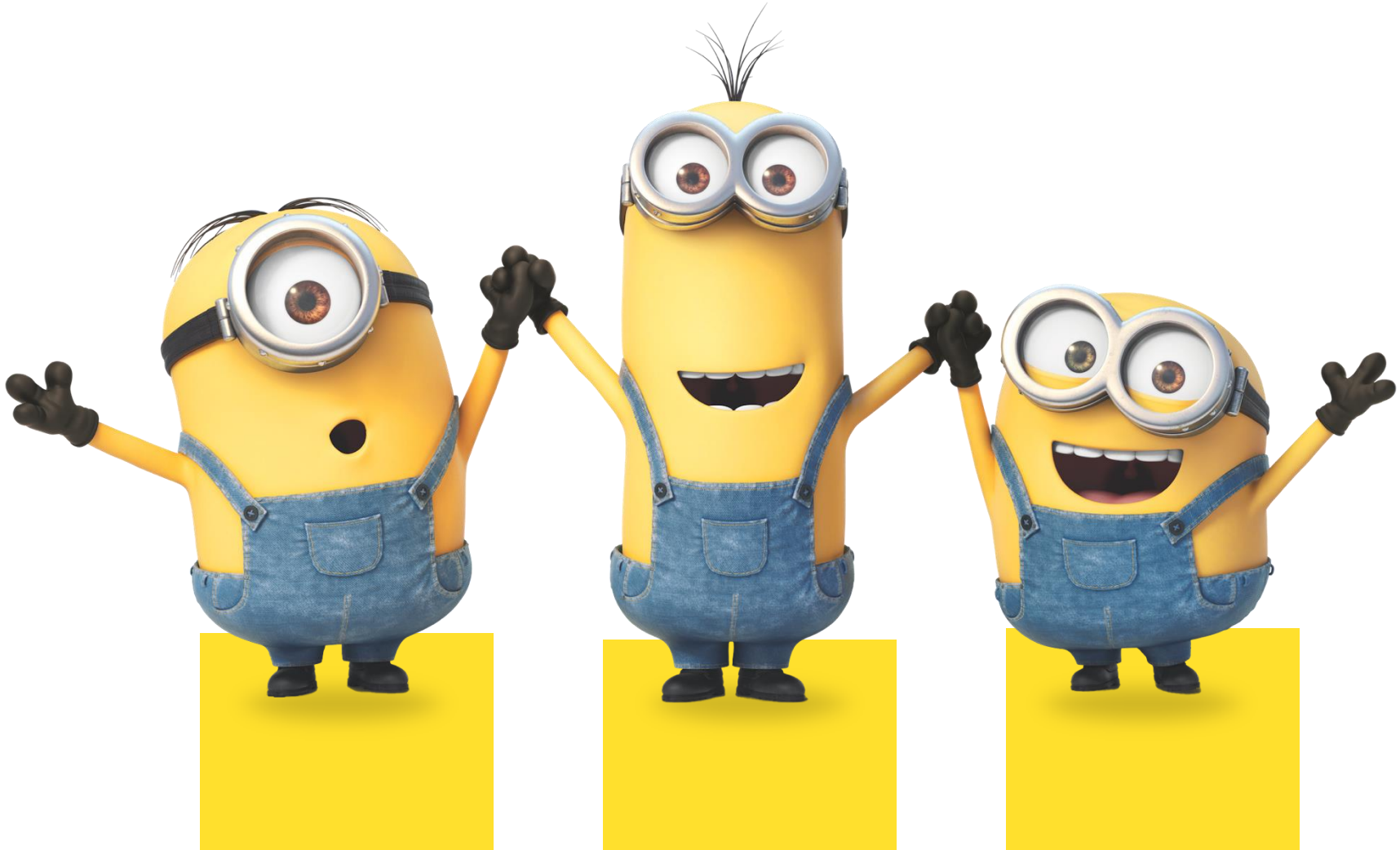
Stuart is known for his one eye and playful nature, making him the prankster of the Minions.

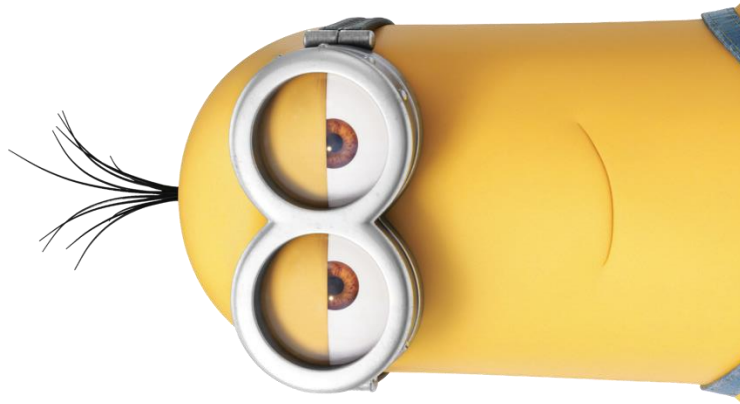
Kevin

As the leadership-driven Minion, Kevin takes charge during their adventures.

Bob

The youngest and most innocent Minion, Bob's enthusiasm often leads to comical situations.



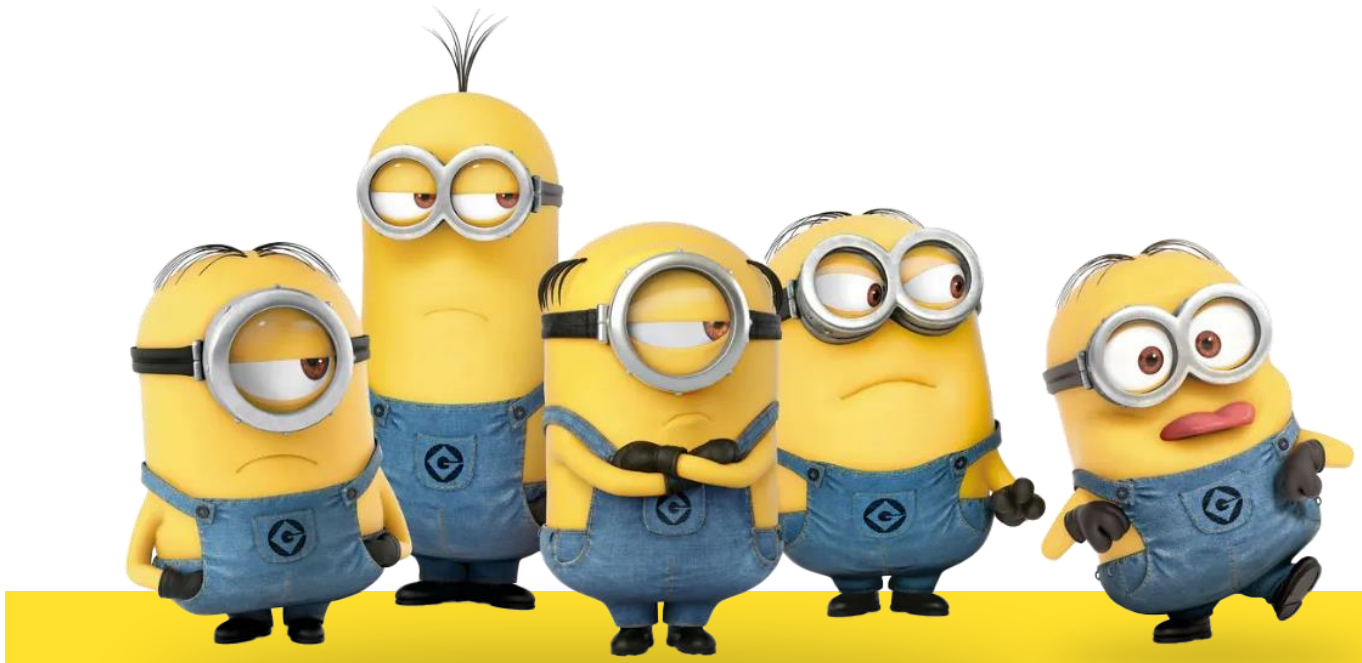


Plot Summary

The "Minions" movie follows the trio of Stuart, Kevin, and Bob on a quest to find a supervillain to serve. Their journey takes them to Villain-Con, where they encounter the diabolical Scarlet Overkill. The Minions' mishaps and humor are central to the plot.

Setting and Locations

The movie features various settings, from the bustling streets of New York City to the grandeur of Villain-Con. The richly detailed locations enhance the storytelling.



Villains and Antagonists

Scarlet Overkill, voiced by Sandra Bullock, is the primary antagonist. Her grand plans and the Minions' hilarious attempts to assist her drive the plot.

Scarlet plans to steal the Imperial State Crown from Queen Elizabeth II.

She promises to reward the Minions if they succeed, but also threatens to kill them if they fail.

Merchandise and Spin-offs

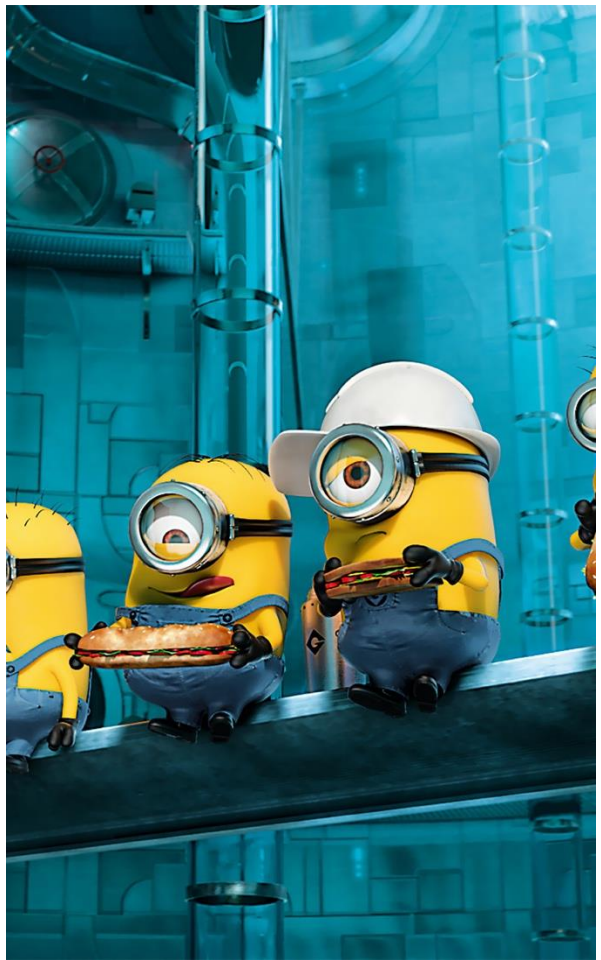
The success of "Minions" has led to a plethora of merchandise, from toys and clothing to video games. Spin-offs like "Minions: The Rise of Gru" expand the franchise.

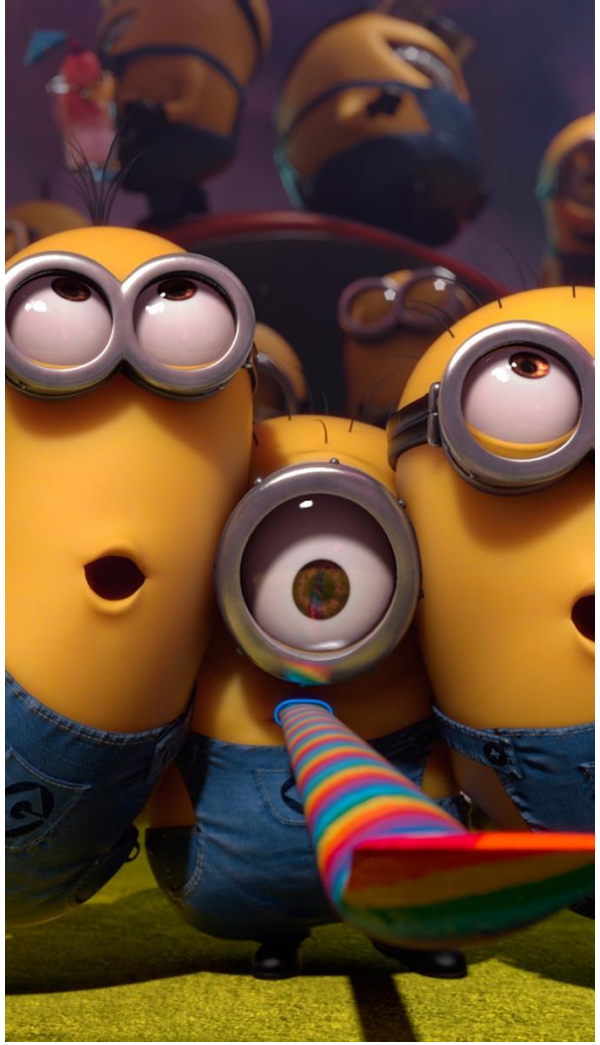
Humour and Gags

Minions is renowned for its slapstick humor, visual gags, and witty dialogue. The Minions' antics and comical situations provide non-stop laughter for audiences of all ages.

Box Office and Awards

The movie features various settings, from the bustling streets of New York City to the grandeur of Villain-Con. The richly detailed locations enhance the storytelling.





$1.159 billion

Worldwide collections

BAFTA Awards, 2016

Best animated film

- **Despicable Me**
- **Despicable Me 2**
- **Minions**
- **Despicable Me 3**
- **Minions: The Rise of Gru**

Sequels and Franchise

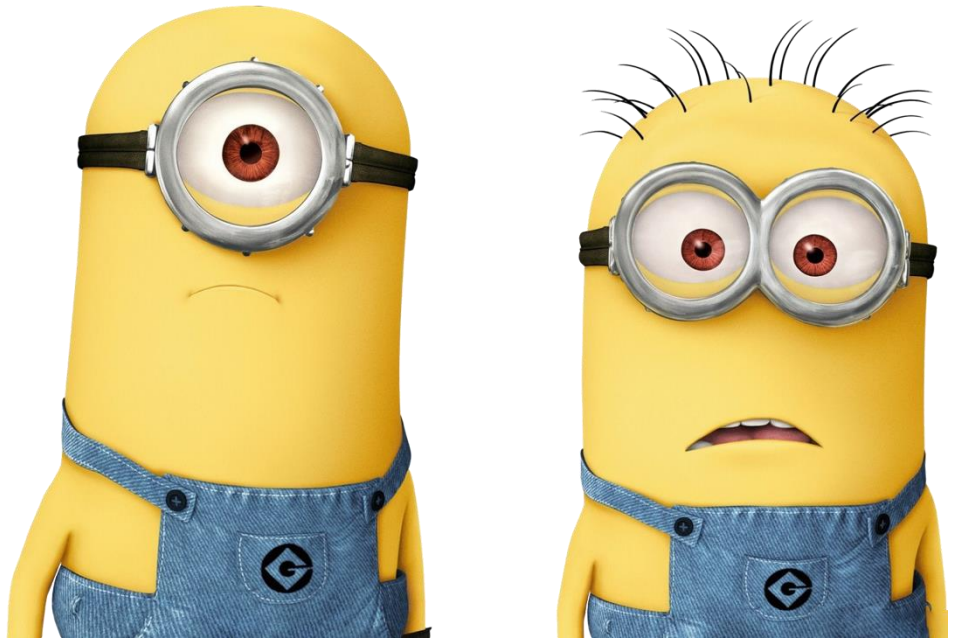
The "Minions" movie is part of the "Despicable Me" franchise, with sequels and spin-offs continuing the adventures of Gru and the Minions. These films explore the origin and further adventures of these beloved characters.

Audience Appeal

"Minions" appeals to a broad audience, captivating children with its humor and entertaining adults with clever references and witty humor.

Cultural Impact

The Minions have transcended their animated origins to become pop culture icons, with appearances in commercials, memes, and even political rallies. Their unique language, Minionese, adds to their charm.



thank-you

Instructions for use

You must credit Slidechef in order to use this template

What you are allowed to do :

Modify this template

Use it for personal or commercial projects

What you are not allowed to do :

Offer Slidechef templates for download

Sublicense, sell or rent any Slidechef content

Distribute Slidechef content unless it has been authorized

Include Slidechef content in an online or offline file

Acquire the copyright of Slidechef content

Slide Chef